

Burchard v. Braunmühl

GTI-Skript

Grundlagen der Theoretischen Informatik

Informatik III

3 Rekursionstheorie

In diesem Kapitel geht es um die Frage, welche Aufgaben prinzipiell von Rechnern bearbeitet werden können. In unserer Sprache: welche Funktionen können von Computern berechnet werden?

Diese Frage ist sehr intuitiv, weil wir nicht gesagt haben, was wir mit Rechnern meinen. Die jetzt handelsüblichen oder die, die erst in der Zukunft geschaffen werden, oder alle prinzipiell möglichen? In diesem Dilemma, das Modell "Rechner" festzulegen bzw. den Begriff "berechnen" zu präzisieren, hat man verschiedene Kalküle entwickelt, von denen man annimmt, daß sie die intuitive Vorstellung gut treffen, und von denen man erwartet, daß sie auch alle zukünftigen Rechnermodelle mit abdecken. Dies ist natürlich nur eine Hoffnung, die aber durch die Tatsache bestärkt wird, daß sich alle vorgeschlagenen Kalküle als äquivalent erwiesen haben, d.h. daß sie alle dieselben Funktionen als berechenbar ausweisen. Dies bedeutet immerhin, daß sich die Menschen heute einigermaßen einig sind, was sie mit dem Begriff "berechenbar" meinen (ob sie auch in Zukunft dieser Meinung bleiben werden, sei dahingestellt). Dies hat den Mathematiker Church zu seiner (Hypo-)These (Church These) veranlaßt, daß der von ihm entwickelte Kalkül - der λ -Kalkül - eine adäquate Beschreibung der berechenbaren Funktionen darstellt. Ebenso hat der Mathematiker Turing die These aufgestellt (Turing-These), daß sein Modell der Turingmaschine eine Präzisierung der intuitiven Vorstellungen von Berechenbarkeit ist.

Nachdem wir jetzt wissen, was berechenbar ist, können wir die Frage noch einmal stellen: welche (arithmetischen) Funktionen sind berechenbar? Um diese Frage zu beantworten, entscheiden wir uns zunächst für einen der vorgeschlagenen Kalküle. Hier haben wir prinzipiell die Wahl zwischen aufwendigen Kalkülen, in denen sich komfortabel Algorithmen entwerfen lassen (in denen man bequem programmieren kann) wie etwa einer der universellen Programmiersprachen, oder einfachen Kalkülen, in denen man leichter Unmöglichkeitsbeweise führen kann, wie etwa, daß es für eine bestimmte Funktion keinen in diesem Kalkül beschreibbaren (implementierbaren) Algorithmus gibt, der sie berechnet. Da wir hier nicht programmieren, sondern über die Frage nachdenken wollen, was im Prinzip berechenbar ist, wählen wir einen der sparsamen Kalküle und unter diesen einen, der eine gewisse Verwandtschaft zu den physikalischen Rechnermodellen hat: wir wählen den Kalkül der Turingmaschine.

Eine Turingmaschine hat ein (potentiell) unendliches Band, das in diskrete Felder eingeteilt ist, die genau ein Zeichen tragen können. Auf dem Band läuft ein Lese-Schreib-Kopf, der von einer Kontrolleinheit gesteuert wird. Die Kontrolleinheit ist ein sequentielles Schaltwerk bzw. ein finiter Automat, kann also endlich viele Zustände einnehmen. Ein Schritt der Turingmaschine besteht darin, das Zeichen eines Feldes zu lesen, und je nach Zustand der Kontrolleinheit, dieses Zeichen durch ein neues auszutauschen und den Kopf ein Feld nach links oder rechts zu rücken oder auch stehen zu lassen. Die Befehle einer Turingmaschine haben also die Form:

(Zustand p , gelesenes Zeichen a , Folgezustand q , zu druckendes Zeichen b , Rückung des Kopfes d)

was wie folgt zu lesen ist: wenn im Zustand p das Bandzeichen a gelesen wird, so wird a durch b ersetzt, der Kopf um d Felder nach rechts ($d \in \{-1, 0, 1\}$) verschoben und in den Zustand q übergegangen.

Wir können uns also eine Turingmaschine als einen finiten Automaten (mit Output) vorstellen mit

einer zusätzlichen Datenstruktur. Input des finiten Automaten ist das gelesene Bandsymbol, Output das zu druckende Bandsymbol und die Rückung des Kopfes (links, rechts, stehenbleiben).

Manchmal ist es auch bequemer sich die Kontrolleinheit als ein kombinatorisches Schaltwerk bzw. eine Schaltfunktion vorzustellen, die als Input das Zustandssymbol und das gelesene Bandsymbol bekommt und als Output das neue Zustandssymbol, das zu druckende Bandsymbol und die Bandrückung liefert.

Wieder andere finden es hilfreicher, auf die Idee des finiten Automaten zu verzichten und sich den Zustand als die Nummer eines Befehls vorzustellen: der Nachfolgezustand ist dann die Nummer des Befehls, der als nächster ausgeführt werden soll. Hier besteht der Befehl (die Anweisung) dann nicht aus einem 5-Tupel wie oben, sondern aus einem if-statement (oder case-statement), das alle 5-Tupel, die mit demselben Zustandszeichen beginnen, zusammenfaßt:

Befehl Nr. p : **case** gelesenes Zeichen = a **then** drucke b , rücke d Felder nach rechts und gehe zu Befehl Nr. q , etc.

3.1 Definitionen

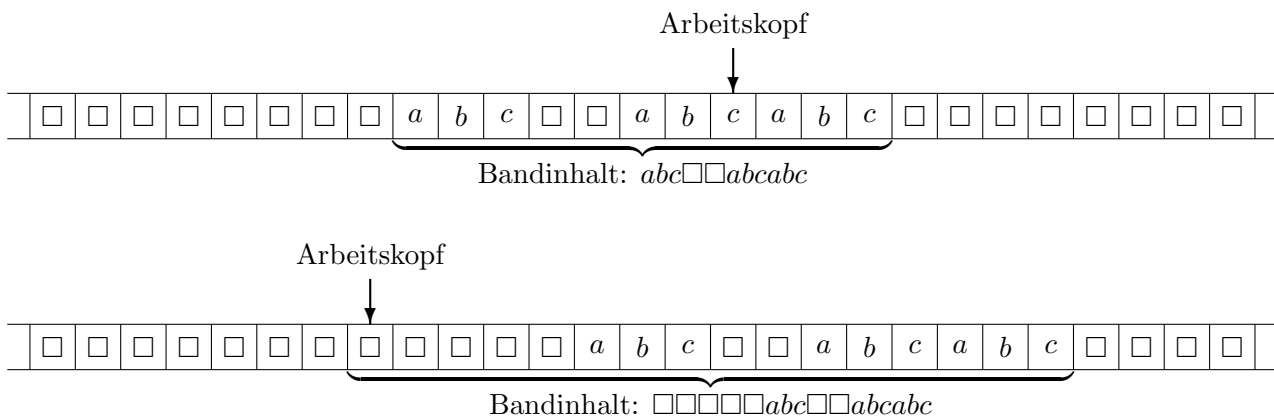
Definition 3.1.1 (Turingmaschine)

Seien Q und Γ endliche Alphabete (Zustandsalphabet und Arbeitsalphabet oder Bandalphabet), $\Sigma \subseteq \Gamma$ (Input- und Outputalphabet), \square ein universelles Sondersymbol nicht aus Γ (Blank, Zeichen für ein leeres Feld), $\Gamma_o = \Gamma \cup \{\square\}$, $\Sigma_o = \Sigma \cup \{\square\}$, $q_0 \in Q$ (Startzustand), $\delta \subseteq Q \times \Gamma_o \times Q \times \Gamma_o \times \{-1, 0, 1\}$ (Menge der Befehle) und $F \subseteq Q$ (Menge der akzeptierenden Zustände), $Q - F$ (Menge der verwerfenden Zustände). Dann ist die Struktur

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

eine *nichtdeterministische Turingmaschine (NTM)*. δ wollen wir auch die Befehlstafel oder das Turingprogramm von M nennen.

Unter dem *Bandinhalt* eines Bandes wollen wir das Wort aus Γ_o^* verstehen, das auf dem kleinsten Teil des Bandes steht, der alle nichtleeren (mit \square bedruckten) Felder und das vom Kopf geprüfte Feld umfaßt.



Die *Situation*, in der sich die Turingmaschine befindet, wird durch ihren Zustand, den Inhalt des Bandes und die Kopfstellung charakterisiert und durch ein Paar $(q, u\#v) \in Q \times \Gamma_o^*\{\#\}\Gamma_o^+$ ($\# \notin \Gamma$) beschrieben, wobei q der Zustand der TM und uv der Bandinhalt ist und der Kopf auf dem ersten Symbol von v steht (das auch \square sein kann). Das Feld, auf dem sich der Kopf in der Situation befindet, nennen wir das *aktuelle Arbeitsfeld* des Bandes und dessen Inhalt das *aktuelle Arbeitssymbol*.

Eine *Startsituation* $s(w)$ bzgl. w hat die Form $(q_0, \# w)$ mit $w \in \Sigma_o^*$. Wir erklären noch den Operator

$$S_{\square} : \Gamma_o^*\{\#\}\Gamma_o^* \longrightarrow \Gamma_o^*\{\#\}\Gamma_o^*$$

vermöge:

$$S_{\square}(u\#v\square) = S_{\square}(\square u\#v) = \begin{cases} u\#v & \text{falls } v \neq \varepsilon \\ u\#\square & \text{sonst} \end{cases} .$$

Die Hülle S_{\square}^* von S_{\square} entfernt alle überflüssigen Blanks (\square), macht also aus jedem Wort aus $\Gamma_o^*\{\#\}\Gamma_o^*$ ein Wort, das ohne das $\#$ einen Bandinhalt darstellt.

Ist ein Befehl (p, b, q, b', d) anwendbar, so geht durch Anwendung dieses Befehls die Situation

$$\begin{aligned} (p, u\#bv) & \text{ über in } (q, S_{\square}^*(ub'\#v)), & \text{ falls } d = 1, v \neq \varepsilon, \\ (p, u\#bv) & \text{ über in } (q, S_{\square}^*(ub'\#\square)), & \text{ falls } d = 1, v = \varepsilon, \\ (p, u\#bv) & \text{ über in } (q, S_{\square}^*(u\#b'v)), & \text{ falls } d = 0, \\ (p, ua\#bv) & \text{ über in } (q, S_{\square}^*(u\#ab'v)), & \text{ falls } d = -1, \\ (p, \#bv) & \text{ über in } (q, S_{\square}^*(\#\square b'v)), & \text{ falls } d = -1, \end{aligned}$$

$$(p \in Q, u, v \in \Gamma_o^*, a, b \in \Gamma_o).$$

Auf der Menge S der Situationen der TM definieren wir die Schrittrelation $\vdash_M \subseteq S \times S$ vermöge: $s \vdash_M s'$, wenn s' aus s durch Anwendung eines Befehls hervorgeht. Die reflexive, transitive Hülle von \vdash_M schreiben wir \vdash_M^* , also $s \vdash_M^* s'$, wenn s' aus s durch Anwendung einer Folge von Befehlen hervorgeht, also wenn es Situationen s_0, \dots, s_n gibt mit $s = s_0$ und $s' = s_n$ und $s_i \vdash_M s_{i+1}$ für alle $i \in \{0, \dots, n-1\}$ ¹. Die Folge s_0, \dots, s_n der durchlaufenen Situationen nennen wir einen *Lauf* der Länge n von s nach s' (der Lauf hat zwar $n + 1$ Situationen, aber es werden nur n Befehle ausgeführt, n Schritte getan).

Die TM M stoppt in einer Situation s , wenn in s kein Befehl anwendbar ist. Wir nennen s dann eine *Stoppsituation*. Ist der Zustand einer Stoppsituation akzeptierend (aus F), so ist s eine *akzeptierende Situation*, andernfalls ist s eine *verwerfende Situation*. Die TM startet immer in einer Situation der Form $(q_0, \# w)$ mit $w \in \Sigma_o^*$. Diese Situation nennen wir auch die zum Input w gehörige *Startsituation* $s(w)$.

Ein Lauf ist *maximal*, wenn er nicht verlängerbar ist (mit einer Stoppsituation endet) oder wenn er unendlich ist. Ein *w-Lauf* ist ein maximaler Lauf, der mit der zu w gehörigen Startsituation $s(w)$ beginnt.

Wir nennen die TM *deterministisch (DTM)*, wenn es keine zwei Befehle aus δ gibt, die in den ersten 2 Symbolen (Zustand, gelesenes Zeichen) übereinstimmen. d.h. für eine deterministische TM gibt es in jeder Situation höchstens einen anwendbaren Befehl. Das bedeutet auch, daß es für jeden Input w genau einen *w-Lauf* gibt.

Die TM M *erkennt* die Sprache $L(M) = \{w \in \Sigma^* \mid s(w) \vdash_M^* s \text{ für eine akzeptierende Stoppsituation } s\}$. Die TM akzeptiert also den Input w , wenn wenigstens ein *w-Lauf* in einer Situation mit akzeptierendem Zustand endet. Wir nennen $L(M)$ auch die Sprache der TM M .

Ist die TM M deterministisch, so ist die von M berechnete *n-stellige Funktion* $f_M^n : (\Sigma^*)^n \longrightarrow \Sigma^*$ mit $f_M^n(u_1, \dots, u_n) = w$ gdw. $(q_0, \#u_1\square \dots \square u_n) \vdash_M^* (q, \# w)$ für ein $q \in F$. Die von der TM M berechnete Funktion ist also auf dem Input definiert, wenn M in einem akzeptierenden Zustand stoppt und der Bandinhalt dann ein Wort aus Σ^* ist und der Kopf auf dem ersten Zeichen dieses Wortes steht. Dieses Wort ist dann der Output bzw. das Bild des Inputs unter der Funktion. In allen anderen Fällen sei die von der TM berechnete Funktion nicht definiert.

Notation 3.1.2

Manchmal ist es bequem, sich die Felder eines Bandes durch \mathbb{Z} nummeriert vorzustellen. Dann wollen wir dem Feld, auf dem der Kopf in der Startsituation steht, die Nummer 1 geben. Damit sind die Nummern der übrigen Felder festgelegt: von links nach rechts in der Ordnung von \mathbb{Z} . Der Input beginnt somit auf Feld 1.

Aufgabe 3.1.3 (DTM=NTM)

Zeigen Sie, daß es zu jeder NTM eine äquivalente DTM gibt (die dieselbe Sprache erkennt bzw. dieselbe

¹Beachte die Verabredung $\{0, \dots, -1\} = \emptyset$

Funktion berechnet).

Idee: bei einer NTM gibt es nicht nur einen w -Lauf für jedes w , sondern viele w -Läufe, die sich in den Situationen baumartig verzweigen können. Den Baum dieser w -Läufe nennen wir w -Berechnungsbaum von M oder Berechnungsbaum für w . Eine deterministische Simulation der nichtdeterministischen Maschine kann darin bestehen, daß jeder dieser w -Läufe nachsimuliert wird. Da es auch unendlich w -Läufe geben kann, empfiehlt es sich, den Baum in der Breitenordnung durchzugehen, d.h. immer längere Anfangstücke der w -Läufe zu simulieren.

Anmerkung 3.1.4

Wir werden in Zukunft, wenn wir von einer Turingmaschine (TM) reden, immer eine deterministische Turingmaschine (DTM) meinen. Wir betrachten also nur deterministische Turingmaschinen, weil es in diesem Zusammenhang keinen Unterschied macht.

Notation 3.1.5

$M(w)$ soll bedeuten: TM M mit Input w (angewandt auf w). $M(w) \downarrow$ heißt: TM M angewandt auf den Input w stoppt nach endlich vielen Schritten und $M(w) \uparrow$: TM M angewandt auf den Input w stoppt nie. $M(w) \downarrow u$ soll heißen: TM M angewandt auf w stoppt mit Output u .

Beispiel 3.1.6

Eine TM, die die Binärdarstellung von n in die von $n + 1$ überführt: Input $\text{bin}(n)$, Output $\text{bin}(n + 1)$ ($\text{bin}(n)$ = Binärdarstellung von n), $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $q_0 = 0$, $F = \{3\}$ und Befehlstafel δ (wir schreiben für $d = -1, 0, 1$ auch L, N, R) :

p	a	p'	a'	d	
0	0	0	0	R	
0	1	0	1	R	laufe über den Input
0	□	1	□	L	
1	0	2	1	L	
1	1	1	0	L	addiere 1 mit Übertrag
1	□	2	1	L	
2	0	2	0	L	
2	1	2	1	L	laufe zum Anfang des Outputs
2	□	3	□	R	

Beispiel 3.1.7

Eine TM, die 1^n in 1^{2n} überführt: Input 1^n , Output 1^{2n} , $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $q_0 = 0$, $F = \{6\}$ und Befehlstafel δ (wir schreiben für $d = -1, 0, 1$ auch L, N, R) :

p	a	p'	a'	d	
0	1	1	□	R	löscht die erste 1
0	□	6	□	R	stoppt
1	1	1	1	R	läuft über das nächste □ rechts
1	□	2	□	R	
2	1	2	1	R	druckt 1 auf das nächste □ rechts
2	□	3	1	R	
3	□	4	1	L	druckt eine (zweite) 1
4	1	4	1	L	läuft über das nächste □ links
4	□	5	□	L	
5	1	5	1	L	läuft vor nächste □ links
5	□	0	□	R	

Definition 3.1.8 (Mehrband-Turingmaschine)

Seien Q und Γ endliche Alphabete (Zustandsalphabet und Arbeitsalphabet), $\Sigma \subseteq \Gamma$ (Inputalphabet), $q_0 \in Q$ (Startzustand), $F \subseteq Q$ (Menge der akzeptierenden Zustände), □ ein Sondersymbol nicht aus Γ (bezeichnet ein leeres Feld) und $\delta \subseteq Q \times \Gamma_o^k \times Q \times \Gamma_o^k \times \{-1, 0, 1\}^k$ (Menge der Befehle).

Dann ist die Struktur

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

eine nichtdeterministische k -Band-Turingmaschine (k -NTM).

Die *Situation*, in der sich die Turingmaschine befindet, wird beschrieben durch $(q, u_1\#v_1, \dots, u_k\#v_k) \in Q \times (\Gamma_o^* \{ \# \} \Gamma_o^+)^k$ ($\# \notin \Gamma$), wobei q der Zustand der TM ist, $u_i v_i$ der Inhalt des i -ten Bandes, wobei der Kopf des i -ten Bandes auf dem ersten Symbol von v_i steht (das auch □ sein kann). Eine *Startsituation* hat die Form $(q_0, \# w, \# \square, \dots, \# \square)$ mit $w \in \Sigma_o^*$.

In der Situation $(p, u_1\#b_1v_1, \dots, u_k\#b_kv_k)$ mit $p \in Q$, $u_i, v_i \in \Gamma_o^*$, $b_i \in \Gamma_o$ ist jeder Befehl anwendbar, der mit (p, b_1, \dots, b_k) beginnt, also jeder Befehl der Form $(p, b_1, \dots, b_k, q, b'_1, \dots, b'_k, d_1, \dots, d_k)$ für gewisse q, b'_1, \dots, b'_k und d_1, \dots, d_k .

Die Menge der Situationen ist $S = Q \times (\Gamma_o^* \{ \# \} \Gamma_o^+)^k$. Die Schrittrelation $\vdash_M \subseteq S \times S$ und die Mehrschrittrelation \vdash_M^* definieren wir analog. Die TM startet immer in einer Situation der Form $(q_0, \# w, \# \square, \dots, \# \square)$ mit $w \in \Sigma^*$. Diese Situation nennen wir auch die zum Input w gehörige *Startsituation* $s(w)$. Lauf, maximalen Lauf, w -Lauf, Stoppsituation, und akzeptierende Stoppsituation definieren wir analog wie in Definition 3.1.1.

Wir nennen die TM *deterministisch* (k -DTM), wenn es keine zwei Befehle aus δ gibt, die im Zustand und in den gelesenen Bandsymbolen übereinstimmen.

Die *Sprache* der TM ist $L(M) = \{w \in \Sigma^* \mid s(w) \vdash_M^* s \text{ für eine akzeptierende Stoppsituation } s\}$. Die TM akzeptiert also den Input w , wenn wenigstens ein w -Lauf in einer Situation mit akzeptierendem Zustand endet.

Ist die TM M deterministisch, so ist die von M berechnete n -stellige Funktion: $f_M^n : (\Sigma^*)^n \rightarrow \Sigma^*$ mit $f_M^n(u_1, \dots, u_n) = w$ gdw. $(q_0, \#u_1\# \dots \#u_n\#, \dots, \#) \vdash_M^* (q, \# w, \#, \dots, \#)$ und $(q, \# w, \#, \dots, \#)$ eine akzeptierende Stoppsituation. Die von der TM M berechnete Funktion ist also auf dem Input definiert, wenn M in einem akzeptierenden Zustand stoppt und der Inhalt von Band 1 dann ein Wort aus Σ^* ist und der Kopf von Band 1 auf dem ersten Zeichen dieses Wortes steht und die übrigen

Bänder leer sind. Dieses Wort ist dann der Output bzw. das Bild des Inputs unter der Funktion. In allen anderen Fällen ist die von der TM berechnete Funktion nicht definiert.

Anmerkung 3.1.9

Auch hier wollen wir in Zukunft nur deterministische k -Band-Turingmaschinen betrachten und sie kurz k -TMen nennen.

3.2 Konstruktion einfacher Turingmaschinen

3.2.1 Spuren

Sei Γ ein endliches Alphabet und M eine Turingmaschine, mit dem Bandalphabet $\Gamma \times \Gamma$, d.h. die Felder des Bandes von M tragen Paare von Zeichen aus Γ . Dann könnten wir bei jedem bedruckten Feld von dem ersten Zeichen und dem zweiten Zeichen aus Γ reden. Sagen wir statt erstes und zweites, oberes und unteres Zeichen, so legt sich die Vorstellung nahe, daß jedes Feld eine obere und eine untere Zelle hat, die jeweils ein Zeichen aus Γ tragen kann. Die Folge der oberen Zellen bilden einen oberen Teil des Bandes - wir sagen eine obere Spur - die Folge der unteren Zellen, eine untere Spur. So können wir uns also das Band in zwei Spuren zerlegt vorstellen, die getrennt beschriftet werden können. Ist der Inhalt eines Feldes $(x, y) \in \Gamma_o^2$, so sagen wir, die Turingmaschine druckt das Zeichen $a \in \Gamma$ in die obere Spur, wenn es auf das Feld das Zeichen (a, y) druckt.

Dieselbe Überlegung kann man natürlich auch mit k Spuren anstellen für ein beliebiges $k \in \mathbb{N}$. Dann ist das neue Bandalphabet Γ_o^k . Man beachte hierbei, daß diese Technik der Alphabetvergrößerung eine Kompression des Bandinhalts auf einen Bruchteil der ursprünglichen Länge erlaubt.

3.2.2 Halbband

Manchmal betrachtet man Turingmaschinen, die nur ein Halbband haben, d.h. deren Kopf nie das Feld links vom Startfeld nach links verläßt. Dies ist keine wirkliche Einschränkung, wie das nächste Lemma sagt:

Satz 3.2.1 (Halbband)

Zu jeder erkennenden Turingmaschine M gibt es eine äquivalente, erkennende Halbband-Turingmaschine M' , die dieselbe Sprache erkennt (dieselbe Funktion berechnet).

Beweis: Ist Γ das Arbeitsalphabet von M , so sei $\Sigma_o \cup \Gamma_o^2 \cup \{\perp\}$ ($\perp \notin \Gamma$) das von der Halbbandmaschine M' , d.h. das Halbband von M' habe zwei Spuren. Zunächst druckt M' die Marke $\perp \notin \Gamma$ auf das Feld links vom Startfeld. Dann arbeitet M' genau wie M , nur auf der oberen Spur. Wenn der Kopf von M das Startfeld nach links verläßt, d.h. die Marke \perp liest, so wechselt M' auf dem Startfeld in die untere Spur und simuliert darauf den linken Teil des Bandes von M allerdings in gespiegelter Felderfolge. Das Band von M wird quasi im Startfeld geknickt, sodaß der linke Teil unter den rechten zu liegen kommt.

Genauer: sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Dann ist $M' = (Q', \Sigma, \Gamma', \delta', 0, F')$. Hierbei ist $Q' = Q \cup \overline{Q} \cup \{0, 1\}$, wo \overline{Q} die Zustände aus Q in gequerrter Version enthält, die Zustände, in denen M' auf der unteren Spur arbeitet, während M' in den Zuständen aus Q auf der oberen Spur arbeitet. 0 sei der neue Startzustand, der noch nicht in $Q \cup \overline{Q}$ vorkomme. $\Gamma' = \Sigma \cup \Gamma_o^2 \cup \{\perp\}$, wobei \perp ein neues Symbol, die Randmarke, sei. $F' = \{q, \bar{q} \mid q \in F\}$. Die Befehlsmenge δ' besteht aus folgenden Befehlen:
 $\{(0, x, 1, x, -1), (1, \square, q_0, \perp, 1)\}$ (M setzt die Marke \perp auf das Feld links vom Startfeld),
 $\{(p, \perp, \bar{p}, \perp, 1), (\bar{p}, \perp, p, \perp, 1) \mid p \in Q, \bar{p} \in \overline{Q}\}$ (M wechselt in die andere Spur),
 $\{(p, x, p, (x, \square), 0), (\bar{p}, x, \bar{p}, (x, \square), 0) \mid x \in \Sigma_o, p \in Q, \bar{p} \in \overline{Q}\}$ (Symbole in die obere Spur) und

$\{(p, (a, x), q, (b, x), d), (\bar{p}, (x, a), \bar{q}, (x, b), -d) \mid x \in \Gamma_o\}$ falls $(p, a, q, b, d) \in \delta$ ($a, b \in \Gamma_o$; $p, q \in Q$). ■

3.2.3 Mehrband

Man betrachtet auch Turingmaschinen mit mehreren Bändern, wie wir sie oben definiert haben. Aber auch hier stellt man fest, daß sie nicht mehr können als Turingmaschinen mit einem Band:

Lemma 3.2.2 (Mehrband)

Zu jeder Mehrband-Turingmaschine M gibt es eine äquivalente Turingmaschine M' mit nur einem Band (die dieselbe Sprache erkennt, dieselbe Funktion berechnet).

Beweis: Sei M eine Turingmaschine mit k Bändern. Wir definieren eine Turingmaschine M' mit einem Band, das aber in $2k$ Spuren unterteilt ist. Ist Γ das Arbeitsalphabet von M , so ist $\Sigma \cup (\{\#, \square\} \times \Gamma_o)^k$ das Arbeitsalphabet von M' , die geraden Spuren tragen Zeichen aus Γ_o , die ungeraden Spuren sind leer bis auf eine einzige Zelle, die ein $\#$ trägt. Das Band i wird durch die Spur $2i$ repräsentiert, d.h. der Inhalt des Bandes i ist auch der Inhalt der Spur $2i$, und das Zeichen $\#$ auf Spur $2i - 1$ steht gerade über dem Feld von Spur $2i$, das dem Arbeitsfeld auf Band i entspricht, auf dem der Kopf steht.

Um einen Befehl von M zu simulieren, läuft der Kopf von M' einmal über das ganze Band und liest die k aktuellen Symbole, über denen ein $\#$ steht, in den Zustand ein, läuft wieder über das Band, ersetzt diese Symbole und verrückt die $\#$ 'e entsprechend den Drucksymbolen und Rückungen des simulierten Befehls. Da jedes noch nicht betretene Feld ein Symbol aus Σ_o trägt, interpretiert M' ein $a \in \Sigma_o$ wie das $2k$ -Tupel, $(\square, a, \square, \dots, \square)$, d.h. für jeden Befehl, der $(\square, a, \square, \dots, \square)$ liest, muß es noch einen geben der a liest, sonst aber gleich ist.

M' benötigt mehr Zeit, da hier ein Kopf das tun muß, was sonst k Köpfe machen. Für jeden Schritt von M überläuft M' den gesamten bedruckten Teil des Bandes zweimal. Arbeitet M also t Schritte lang, so kann M' bis zu $2t^2$ Simulationsschritte ausführen (der Inhalt eines Bandes von M kann höchstens t Felder lang werden). ■

3.2.4 Schaltungen

Definition 3.2.3 (Serienschaltung)

Seien $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $M' = (Q', \Sigma, \Gamma', \delta', q'_0, F')$ Turingmaschinen mit Q, Q' disjunkt und $\Gamma \subseteq \Gamma'$. Dann definieren wir eine Serienschaltung $M \circ M' = (Q \cup Q', \Sigma, \Gamma', \delta'', q_0, F')$, mit

$$\delta'' = \delta \cup \delta' \cup \{(q, x, q'_0, x, 0) \mid q \in F, x \in \Gamma_o, \text{ für kein } q', y, d \text{ ist } (q, x, q', y, d) \in \delta\}.$$

(immer wenn M in einem akzeptierenden Zustand stoppt, läuft M' an).

Definition 3.2.4 (Aufruf von Turingmaschinen)

Seien $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $M' = (Q', \Sigma, \Gamma, \delta', q'_0, F')$ Turingmaschinen mit Q, Q' disjunkt, $p_A, p_R \in Q$. Dann definieren wir die Turingmaschine $M(M') = (Q \cup Q', \Sigma, \Gamma, \delta'', q_0, F)$, mit

$$\begin{aligned} \delta &= \delta \cup \delta' \cup \{(p_A, x, q'_0, x, 0) \mid x \in \Gamma_o\} \\ &\quad \cup \{(q', x, p_R, x, 0) \mid q' \in F', x \in \Gamma_o, \text{ für kein } p, y, d \text{ ist } (q', x, p, y, d) \in \delta'\} \end{aligned}$$

(im Zustand p_A ruft M die TM M' auf. Stoppt M' akzeptierend, so springt die Rechnung in den Zustand p_R von M zurück.)

3.2.5 Abzählbar, aufzählbar, erkennbar, entscheidbar

Nun definieren wir die Eigenschaften von Sprachen, die uns in diesem Kapitel am meisten interessieren: Wir können die Elemente einer Mengen A durchnummerieren, sodaß wir vom ersten, zweiten, dritten Element usw. reden können. Wichtig daran ist, daß jedes Element der Menge eine eigene Nummer bekommt. Wenn wir jedem Element genau eine Nummer geben, so haben wir eine bijektive Abbildung zwischen den natürlichen Zahlen aus \mathbb{N} und den Elementen der Menge A definiert. Wenn uns dies gelingt, so nennen wir die Menge A abzählbar unendlich (wir können die Elemente von A durchzählen oder abzählen). Die endlichen Mengen können wir natürlich durchnummerieren, aber wir werden dafür nicht alle natürlichen Zahlen verbrauchen. So können wir eine Menge A abzählbar nennen, wenn es eine Bijektion zwischen \mathbb{N} und A gibt oder A endlich ist. Um beide Fälle unter einen Hut zu bringen, erlauben wir, daß ein Element aus A auch mehrmals aufgeführt wird, also mehrere Nummern bekommt. Dann können wir die natürlichen Zahlen auf endlich viele Elemente verteilen und so aufbrauchen. Damit hätten wir also eine totale, surjektive Abbildung von \mathbb{N} auf A definiert.

Wenn das Ganze auf berechenbare Weise geht, wenn also die totale, surjektive Abbildung zwischen \mathbb{N} und A auch noch durch eine Turingmaschine berechenbar ist, so nennen wir A rekursiv aufzählbar (hier müssen wir freilich die Elemente von \mathbb{N} durch ihre unäre Codierung aus $\{1\}^*$ ersetzen und A muß eine Wortmenge (Sprache) sein).

Eine Sprache A heißt erkennbar, wenn sie durch eine Turingmaschine erkannt wird, d.h. wenn es eine Turingmaschine gibt, die genau auf den Worten aus A akzeptierend stoppt. Für Worte, die nicht aus A sind, muß die Turingmaschine nicht notwendig stoppen (und somit eine Antwort geben). Gibt es für eine Sprache A eine Turingmaschine, die sie erkennt und dabei immer (d.h. auf jedem Input) stoppt, so nennen wir A entscheidbar.

Aus einer Turingmaschine M , die A erkennt, kann man immer eine Turingmaschine M' machen, die eine partielle Funktion $\chi : \Sigma^* \rightarrow \{0, 1\}$ berechnet: M' arbeitet genau wie M , nur, stoppt M in einem akzeptierenden Zustand, so gibt M' als Output 1 aus, stoppt M in einem verwerfenden Zustand, so so gibt M' als Output 0 aus. Stoppt M nicht, so auch M' nicht. Für die von M' berechnete partielle Funktion $\chi : \Sigma^* \rightarrow \{0, 1\}$ gilt offenbar: $A = \chi^{-1}(1)$, d.h. genau die Worte aus A werden in die 1 abgebildet. Allerdings gibt es Worte, für die χ nicht definiert ist.

Wenn diese Funktion χ total ist, d.h. wenn die Turingmaschine M' immer stoppt, so ist χ die charakteristische Funktion von A . Offenbar ist A genau dann entscheidbar, wenn ihre charakteristische Funktion berechenbar ist.

Anmerkung 3.2.5

Eine Funktion $f : A \rightarrow B$ heißt *bijektiv*, wenn sie total, surjektiv und injektiv ist, d.h. wenn es für jedes $a \in A$ genau ein $b \in B$ und für jedes $b \in B$ genau ein $a \in A$ gibt, mit $f(a) = b$.

Definition 3.2.6 (abzählbar, aufzählbar, erkennbar, entscheidbar)

Sei $L \subseteq \Sigma^*$. Wir definieren:

1. L ist *abzählbar* (engl. denumerable) gdw.
 L endlich oder es gibt eine bijektive Funktion $a : \mathbb{N} \rightarrow L$
 (es gibt eine Auflistung aller Worte aus L , in der jedes Wort aus L genau einmal vorkommt).
2. L ist (*rekursiv*) *aufzählbar* (engl. recursive enumerable) gdw.
 L endlich oder es gibt eine bijektive, berechenbare Funktion: $\{1\}^* \rightarrow L$.
 (die Auflistung kann von einer TM vorgenommen werden).
3. L ist *erkennbar* (engl. recognizable) gdw. es gibt eine TM M , die L erkennt.
4. L ist (*rekursiv*) *entscheidbar* (engl. recursive) gdw.
 es gibt eine TM M , die L erkennt und für jeden Input aus Σ^* stoppt, oder äquivalent:
 es gibt eine totale, berechenbare Funktion: $\chi : \Sigma^* \rightarrow \{0, 1\}$ mit $L = \chi^{-1}(1)$.

Aufgabe 3.2.7

Zeige:

1. Ist $L \neq \emptyset$ und gibt es eine totale, surjektive Funktion $a : \mathbb{N} \rightarrow L$, dann ist L abzählbar.
2. Ist $L \neq \emptyset$ und gibt es eine totale, surjektive, berechenbare Funktion $a : \mathbb{N} \rightarrow L$, dann ist L rekursiv aufzählbar.

Anmerkung 3.2.8

O.B.d.A. können wir immer annehmen, daß ein Alphabet ein Anfangsabschnitt $\{0, \dots, n\}$ der natürlichen Zahlen ist. Wir ordnen die Symbole, nummerieren sie in ihrer Reihenfolge durch und identifizieren sie dann mit ihren Nummern.

Lemma 3.2.9

Ist Σ endlich, so ist Σ^* aufzählbar.

Beweis: Sei $\Sigma = \{0, \dots, p - 1\}$ mit der kanonischen Ordnung der Zahlen. Wir definieren folgende Reihenfolge auf Σ^* :

1. Ordne die Worte aus Σ^* der Länge nach.
2. Ordne die Worte gleicher Länge "alphabetisch" in der Ordnung von Σ .

Sei $w : \mathbb{N} \rightarrow \Sigma^*$ die dadurch induzierte bijektive Aufzählung ($w(i) =$ das i -te Wort in dieser Reihenfolge). Wir können uns ein Wort über Σ auch als eine p -al-Darstellung (p -näre Darstellung) der natürlichen Zahlen (mit führenden Nullen) vorstellen. Dann wäre die Nummer eines Wortes w gerade $i = z(w) + s(|w|)$, wo $z(w)$ die Zahl mit der p -nären Darstellung w sei und $s(k) = \sum_{i=0}^{k-1} p^i = |\Sigma^{<k}|$ (0^n bekommt die Nummer $s(n)$ und alle Worte w der Länge n die Nummer $s(n) + z(w)$).

Wir definieren eine TM M , die w berechnet. M hat 2 Bänder. Der Input 1^i steht auf dem ersten Band.

1. Liest M auf Band 1 eine 1, so rückt M auf Band 1 ein Feld nach rechts, geht zu (2). Liest M auf Band 1 ein \square , so gehe zu (3).
2. M addiert auf Band 2 eine 1 p -när. Läuft der Überlauf dabei bis vor den Bandinhalt, so ersetzt M alle Symbole auf Band 2 durch 0 und setzt auf das \square davor eine 0, gehe zu (1).
3. M löscht Band 1, kopiert Band 2 auf Band 1, setzt den Kopf von Band 1 auf das erste Zeichen, löscht Band 2 und stoppt. ■

Definition 3.2.10

Diese Aufzählung der Worte von Σ^* , wollen wir die *kanonische Aufzählung* von Σ^* nennen. Wir nennen eine *Ordnung* (Σ^*, \leq) *rekursiv*, wenn die dadurch induzierte Bijektion $w : \mathbb{N} \rightarrow \Sigma^*$ vermöge $w(0) =$ das kleinste Wort aus Σ^* , $w(i) =$ das kleinste Wort größer als $w(i - 1)$ berechenbar ist.

Aufgabe 3.2.11

Zeigen sie, daß folgende Aussagen äquivalent sind:

1. L aufzählbar.
2. es gibt eine aufzählbare Menge D und eine totale, surjektive, berechenbare Funktion: $D \rightarrow L$.
3. es gibt ein Alphabet Σ und eine totale, surjektive, berechenbare Funktion: $\Sigma^* \rightarrow L$.

Aufgabe 3.2.12

L erkennbar gdw. es existiert eine berechenbare Funktion f mit $L = \text{Def } f$. ($\text{Def } f =$ Definitionsbereich f).

Aufgabe 3.2.13

L ist aufzählbar gdw. L ist Bild einer berechenbaren Funktion f .

Aufgabe 3.2.14

L entscheidbar $\implies \bar{L}$ entscheidbar.

Definition 3.2.15

L heißt *monoton aufzählbar* (bezüglich einer rekursiven Ordnung \leq auf Σ^*), wenn es eine totale, surjektive, monotone und berechenbare Aufzählung $a : \mathbb{N} \rightarrow L$ gibt (d.h. $i \leq j \implies a(i) \leq a(j)$).

Satz 3.2.16 (monoton aufzählbar)

L monoton aufzählbar (bzgl. einer rekursiven Ordnung \leq auf Σ^*) gdw. L entscheidbar.

Beweis: Sei (Σ^*, \leq) eine rekursive Ordnung auf Σ^* vermöge einer Bijektion $w : \mathbb{N} \rightarrow \Sigma^*$ berechenbar durch die TM M_w .

\implies : L sei monoton aufzählbar durch $a : \mathbb{N} \rightarrow L$, und M_a eine TM, die a berechnet. Ist L endlich, so ist L entscheidbar. Sei L also unendlich. Dann wird L durch folgende TM M entschieden: ist $u \in \Sigma^*$, so simuliert $M(u)$ die TM $M_a(i)$ für $i = 1, 2, 3, \dots$ bis ein Output u ergibt oder ein Wort, das größer als u ist (in der Ordnung auf Σ^*). Im ersten Fall stoppt $M(u)$ in einem akzeptierenden Zustand, im zweiten Fall in einem verwerfenden:

$M(u)$ arbeitet genauer wie folgt.

```
i:=0;
while a(i) < u do begin i:=i+1;
if a(i)=u then Stop akzeptierend else Stop verwerfend;
```

Noch genauer: M hat 3 Bänder, der Input steht zu Anfang auf Band 1.

1. M kopiert Band 2 auf Band 3, wendet M_a auf Band 3 an und druckt eine 1 hinter Bandinhalt 2, (2)
2. M vergleicht $u = \text{Bandinhalt 1}$ mit $W = \text{Bandinhalt 3}$:
 ist $u = W$, so stoppt M akzeptierend.
 ist $u < W$, so stoppt M verwerfend
 sonst (1).

\Leftarrow : Ist L endlich, so ist dies klar. Sei also L unendlich und entscheidbar durch die TM M_L . Definiere eine monotone Aufzählung $a : \mathbb{N} \rightarrow L$ vermöge:

$$\begin{aligned} a(0) &= \min\{w \mid \chi(w) = 1\}, \\ a(i) &= \min\{w \mid \chi(w) = 1 \text{ und } w > a(i-1)\}, \end{aligned}$$

Da L entscheidbar, also die charakteristische Funktion χ von L berechenbar, ist auch a berechenbar. a wird genauer wie folgt berechnet:

Sei $w : \mathbb{N} \rightarrow \Sigma^*$ die durch die Ordnung \leq induzierte Aufzählung: $M_a(1^k)$ arbeitet wie folgt (z ist die Zahl der Treffer, beim $-$ ten hören wir auf):

```
z:=0;
for i = 0 to  $\infty$  do begin %(das i-te Wort in  $\Sigma^*$  wird überprüft)
if  $M_L(w(i))$  stoppt in akzeptierendem Zustand then begin
if z=k then return(w(i));
z := z + 1;
endif
endfor
```

Noch genauer: M_a hat 5 Bänder, der Input steht zu Anfang auf Band 1. Die übrigen Bänder sind leer.

1. M_a kopiert Band 3 auf Band 4, druckt eine 1 hinter Bandinhalt 3,

- wendet M_w auf Band 4 an und kopiert den Output auf Band 5, (2).
2. M_a wendet M_L auf Band 5 an: stoppt M_L akzeptierend, (3), stoppt M_L verwerfend, so löscht M_a Band 4 und 5, (1)
 3. M_a vergleicht die Inhalte von Band 1 und Band 2. Sind sie gleich, so (4), sonst löscht M_a Band 4 und 5 und druckt eine 1 hinter Bandinhalt 2, (1).
 4. M_a löscht Band 1, kopiert Band 4 auf Band 1, löscht Band 2-5 und stoppt akzeptierend.

■

3.2.6 Dove-Tailing

Dove-tailing (dove Schwalbe, tail Schwanz) ist eine Zimmermannstechnik zum Verzahnen von Brettern durch Fugen von der Art von Schwalbenschwänzen. So nennen wir auch die Technik, zwei Aufzählungen zu verzahnen, dove-tailing. Im folgenden Satz wird eine Aufzählung von Σ^* verzahnt mit der Aufzählung der Schrittzahlen einer simulierten TM.

Satz 3.2.17 (erkennbar = aufzählbar)

$L \subseteq \Sigma^*$ ist erkennbar, gdw. L aufzählbar ist.

Beweis: Ist L endlich, so ist der Satz klar. Sei also L unendlich.

1) Sei L erkennbar und M_L die TM, die L erkennt. Wir konstruieren eine Turingmaschine M_a , die L aufzählt. Sei M_w eine TM, die Σ^* aufzählt. Die TM M_a geht (mithilfe von M_w) alle Worte von Σ^* durch und überprüft, ob sie von M_L erkannt werden. Immer wenn M_L erkennt, nimmt M_a das aktuelle Wort in seine Aufzählung auf. Allerdings kann M_a die TM M_L nicht beliebig lange laufen lassen, da M_a sonst beim ersten Wort, auf dem M_L unendlich lange läuft, hängen bleiben würde. Darum wird hier die Technik des dove-tailing verwandt: M_L läuft auf dem ersten Wort nur einen Schritt, dann auf den ersten beiden Worten zwei Schritte, auf den ersten drei drei Schritte, allgemein auf den ersten k Worten k Schritte, und dies nacheinander für alle $k \in \mathbb{N}$.

Damit kann die TM M_L nie bei einem Wort hängen bleiben, da sie immer nur endliche Zeit auf einem Input läuft, wird aber letztlich doch auf jedem Wort jede gewünschte Schrittzahl laufen. Bei diesem Prozeß wird M_L immer wieder einen Input akzeptieren. In unsere Aufzählung werden diese Inputs in der Reihenfolge aufgenommen, in der sie akzeptiert werden.

Genauer: $M_a(l)$ arbeitet wie folgt: (j ist die Zahl der Treffer, k wie oben)

```

j := 0; S := ∅;
for k = 0 to ∞ do begin
  for i = 0 to k do begin
    if  $M_L(w(i))$  stoppt in k Schritten and  $i \notin S$  then begin
      if j = 1 then Return (w(i));
      j := j + 1;
      S := S ∪ {i};
    endif
  endfor
endfor
endfor

```

2) Sei L aufzählbar und M_a die TM, die L aufzählt. Wir konstruieren eine TM M_L , die L erkennt. Falls das Wort $w \in L$, wird M_a für irgendein $i \in \mathbb{N}$ dieses w ausgeben. $M_L(w)$ läßt also M_a für alle Zahlen $i = 0, 1, 2, \dots$ laufen, bis einmal w ausgegeben wird. Ist $w \notin L$, so wird M_L allerdings nicht stoppen, d.h. M_L stoppt genau auf den Worten aus L .

Genauer: $M_L(w)$ arbeitet wie folgt:

$i=0$; while $M_a(1^i) \neq w$ do $i := i + 1$. ■

3.2.7 Parallelschaltung

Wir können zwei Turingmaschinen M_1 und M_2 nicht nur in Serie schalten, sondern auch parallel laufen lassen. Eine solche Parallelschaltung $[M_1 \parallel M_2]$ ist eine 2-Band-TM, die abwechselnd die TM M_1 einen Schritt auf dem ersten Band und dann die TM M_2 einen Schritt auf dem zweiten Band simuliert. Sie stoppt, wenn eine der beiden Maschinen zum Halten kommt.

Satz 3.2.18

$L \subseteq \Sigma^*$ ist entscheidbar, gdw. L und \bar{L} erkennbar sind.

Beweis:

1) Ist L entscheidbar durch die TM M , so ist L auch erkennbar durch M . \bar{L} wird durch dieselbe TM M erkannt, wenn man nur akzeptierende und verwerfende Zustände vertauscht.

2) Sei L erkennbar durch die TM M und \bar{L} durch die TM \bar{M} . Wir konstruieren eine Turingmaschine $M_e = [M \parallel \bar{M}]$, die L entscheidet. $M_e(w)$ läßt die TMen M und \bar{M} parallel laufen. Genau eine von beiden wird w akzeptieren. M_e stoppt in einem akzeptierenden Zustand, wenn M akzeptiert und in einem verwerfenden, wenn \bar{M} akzeptiert.

Genauer:

1. kopiere Band 1 auf Band 2, (2).
2. wende M einen Schritt auf Band 1 an und \bar{M} auf Band 2 an, (3).
3. stoppt M , so stoppe in akzeptierendem Zustand gdw. M akzeptiert. Sonst (4).
4. stoppt \bar{M} , so stoppe in akzeptierendem Zustand gdw. \bar{M} akzeptiert. Sonst (2).

Noch genauer:

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $\bar{M} = (Q', \Sigma, \Gamma', \delta', q'_0, F')$. Dann sei

$$M_e = M \times \bar{M} = (\{1, 2\} \cup \{1, 2\} \times Q \times Q', \Sigma, \Gamma \cup \Gamma', \delta_e, 1, \{1\} \times F \times Q' \cup \{2\} \times Q \times (Q' - F'))$$

mit

$$\begin{aligned} (p, A, p', A', d) \in \delta, \text{ so } \{((1, p, q), (A, X), (2, p', q), (A', X), (d, 0)) \mid q \in Q', X \in \Gamma'\} \subseteq \delta_e \\ (q, B, q', B', d) \in \delta', \text{ so } \{((2, p, q), (X, B), (1, p, q'), (X, B'), (0, d)) \mid p \in Q, X \in \Gamma\} \subseteq \delta_e \end{aligned}$$

und außerdem seien noch für alle $a \in \Sigma$ folgende Befehle aus δ_e (die zu Anfang Band 1 auf Band 2 kopieren):

$$\begin{aligned} (1, (a, \square), 1, (a, a), (1, 1)) \\ (1, (\square, \square), 2, (\square, \square), (-1, -1)) \\ (2, (a, a), 2, (a, a), (-1, -1)) \\ (2, (\square, \square), (1, q_0, q'_0), (\square, \square), (1, 1)) \end{aligned}$$

Korollar 3.2.19

$L \subseteq \Sigma^*$ ist entscheidbar, gdw. L und \bar{L} aufzählbar sind. ■

3.3 Anzahl der berechenbaren Funktionen

Wollen wir Turingmaschinen selbst zum Objekt von Berechnungen machen, so müssen wir sie als Worte codieren. Dazu müssen wir i.w. nur die Befehle hintereinanderschreiben, die Zustände, die Arbeitssymbole, Anfangszustand und akzeptierende Zustände. Da jede Turingmaschine ihre eigenen Alphabete hat, der Code aber über einem festgelegten Alphabet definiert sein muß, müssen wir auch die Symbole der einzelnen Turingmaschinen codieren. Natürlich gibt es vielerlei Arten, Turingmaschinen als Worte niederzuschreiben, wir müssen uns nur auf eine festlegen. Wir wollen hier als Codealphabet $\{0, 1\}$ wählen, alle Zeichen von Turingmaschinenalphabeten durch 1-Strings codieren und die 0 als Trennzeichen benützen.

Definition 3.3.1 (Turingprogramm)

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine TM. O.B.d.A. seien alle Alphabete Anfangsstücke von \mathbb{N}_+ ($Q = \{1, \dots, |Q|\}$, etc.). Die Null 0 sei das Zeichen \square für das leere Feld. Sei $\text{un}: \mathbb{N} \rightarrow \{1\}^*$ die Funktion, die jeder natürlichen Zahl ihre Unärdarstellung zuordnet, also $\text{un}(n) = 1^n$. Dann codieren wir einen Befehl aus δ :

$$\text{cod} : (p, a, q, b, d) \mapsto \text{un}(p)0 \text{un}(a)0 \text{un}(q)0 \text{un}(b)0 \text{un}(d).$$

Sei $\delta = \{B_1, \dots, B_k\}$, (B_i die Befehle der TM). Dann codieren wir δ durch:

$$\text{cod} : \delta \mapsto \text{cod}(B_1)0 \cdots 0 \text{cod}(B_k).$$

Sei $F = \{p_1, \dots, p_l\}$, dann ist codieren wir die Menge der akzeptierenden Zustände F durch:

$$\text{cod} : F \mapsto \text{un}(p_1)0 \cdots 0 \text{un}(p_l).$$

Schließlich codieren wir die ganze Turingmaschine M durch:

$$\text{cod} : M \mapsto \text{un}(|Q|)00 \text{un}(|\Sigma|)00 \text{un}(|\Gamma|)00 \text{cod}(\delta)00 \text{un}(q_0)00 \text{cod}(F)$$

Das Wort $\text{cod}(M)$ über $\{0, 1\}^*$ nennen wir ein *Turingprogramm* von M .

Jedes Wort aus $\Sigma^* \supseteq \{0, 1\}^*$, das nicht Turingprogramm einer TM ist, erklären wir als ein Turingprogramm der *leeren TM* $M_\emptyset = (\{0\}, \Sigma, \Sigma, \emptyset, 0, \{0\})$, die keinen Befehl hat. Damit ist jedes Wort aus Σ^* auch Turingprogramm. Die Turingmaschine mit dem Turingprogramm u bezeichnen wir mit T_u .

Definition 3.3.2

Jede Nummerierung von Σ^* führt zu einer *Nummerierung der Turingmaschinen*: ist u das i -te Wort, so nennen wir T_u die i -te TM. (Man kann sich u auch als Repräsentation der Zahl i vorstellen. u wird daher auch oft der Index der Turingmaschine T_u genannt).

Dies ist der Ort, wo wir eine erste Antwort liefern können auf die Frage, welche Probleme, Funktionen, Sprachen von Turingmaschinen behandelt werden können. Da jede Turingmaschine höchstens eine Funktion berechnet und jedes Turingprogramm höchstens eine Turingmaschine codiert, kann die Zahl der berechenbaren Funktionen nicht größer sein als die Zahl der Turingprogramme, bzw. die Zahl der Worte aus Σ^* . Wie wir schon gesehen haben, ist das gerade die Zahl der natürlichen Zahlen aus \mathbb{N} , denn wir können eine Bijektion zwischen Σ^* und \mathbb{N} angeben. Die Zahl der arithmetischen Funktionen von \mathbb{N} nach \mathbb{N} , ja sogar die Zahl der zweiwertigen Funktionen von \mathbb{N} nach $\{0, 1\}$, ist bei weitem größer als die Zahl der natürlichen Zahlen. Die Menge dieser Funktionen schreibt man suggestiv auch $2^{\mathbb{N}}$, was schon andeutet, daß es "exponentiell" mehr solche Funktionen als natürliche Zahlen gibt. Damit ist schon klar, daß die allermeisten arithmetischen Funktionen nicht berechenbar sind.

Notation 3.3.3

$$\begin{aligned} \{0, 1\}^{\mathbb{N}} &= \text{Menge der totalen Funktionen von } \mathbb{N} \text{ in } \{0, 1\}, \\ &= \text{Menge der unendlichen Tupel über } \{0, 1\}, \\ &= \text{Menge der unendlichen Worte über } \{0, 1\}. \end{aligned}$$

Anmerkung 3.3.4

Statt $\{0, 1\}^{\mathbb{N}}$ findet man in vielen Büchern auch $\{0, 1\}^\omega$ vor allem, wenn man damit die Menge der unendlichen Worte über $\{0, 1\}$ meint. Formal gesehen sind die oben genannten Mengen alle dieselben.

Satz 3.3.5

$\{0, 1\}^{\mathbb{N}}$ ist nicht abzählbar.

Beweis: Wäre die Menge der unendlichen Tupel über $\{0, 1\}$ abzählbar, so gäbe es eine Bijektion $\mathbb{N} \longleftrightarrow \{0, 1\}^{\mathbb{N}}$, d.h. eine Nummerierung der unendlichen Tupel

$$\begin{array}{rcccc} t_0 & = & t_0^0 & t_0^1 & t_0^2 & \cdots \\ t_1 & = & t_1^0 & t_1^1 & t_1^2 & \cdots \\ t_2 & = & t_2^0 & t_2^1 & t_2^2 & \cdots \\ \vdots & & \vdots & \vdots & \vdots & \vdots \\ t_i & = & t_i^0 & t_i^1 & t_i^2 & \cdots \\ \vdots & = & \vdots & \vdots & \vdots & \vdots \end{array}$$

in der jedes unendliche Tupel genau einmal vorkommt. Sei $t = t_0^0 t_1^1 t_2^2 t_3^3 \cdots$ das Tupel der Diagonalelemente der obigen Matrix und $\bar{t} = \bar{t}_0^0 \bar{t}_1^1 \bar{t}_2^2 \bar{t}_3^3 \cdots$ das Tupel der invertierten Diagonalelemente mit $\bar{t}_i^i = 0$ wenn $t_i^i = 1$ und $\bar{t}_i^i = 1$ wenn $t_i^i = 0$. Das Tupel \bar{t} kommt in der Nummerierung nicht vor, denn wäre $\bar{t} = t_k$ für ein $k \in \mathbb{N}$, so wäre $t_k^k = \bar{t}_k^k$ W! Damit ist gezeigt: Es kann keine Nummerierung von $\{0, 1\}^{\mathbb{N}}$ geben, die jedes Tupel aufführt. ■

3.4 Das Halteproblem

Wir definieren jetzt zwei interessante Sprachen, die unter den von Turingmaschinen erkennbaren Sprachen eine besondere Rolle spielen:

Definition 3.4.1 (Halteproblem, Selbstanwendbarkeitsproblem)

$$\begin{array}{ll} H = \{x\#y \mid x, y \in \Sigma^* \text{ mit } T_x(y) \downarrow\} & \text{das Halteproblem von Turingmaschinen } (\# \notin \Sigma), \\ S = \{x \mid x \in \Sigma^* \text{ mit } T_x(x) \downarrow\} & \text{das Selbstanwendbarkeitsproblem von Turingmaschinen.} \end{array}$$

Anmerkung 3.4.2

Die Sprache H ist über dem Alphabet $\Sigma_{\#} = \Sigma \cup \{\#\}$ gebildet, d.h. wir haben damit das der Theorie zugrundegelegte Inputalphabet Σ verlassen. Schöner und stimmiger wäre es, würden wir auch mehrstellige Sprachen betrachten, d.h. Wortetupel aus $\Sigma \times \cdots \times \Sigma$. Dann könnten wir statt $x\#y$ besser $(x, y) \in \Sigma \times \Sigma$ schreiben, d.h. auf dem Band der TM stünde der Input $x \square y$. H wäre dann eine zweistellige Sprache über dem Alphabet Σ . Dennoch wollen wir in dieser Vorlesung bei den üblichen Vorstellungen bleiben.

Satz 3.4.3

Die Mengen H und S sind erkennbar.

Beweis: Wir definieren eine Turingmaschine U , die bei Input $x\#w$ das Turingprogramm x auf den Input w anwendet. U hat 3 Bänder.

1. U kopiert x von Band 1 auf Band 2 und löscht dabei x von Band 1. U druckt den Startzustandscode $1^q 0$ aus x vor das $\#$ auf Band 3, (2).
2. Steht auf Band 3 der String $1^p 0$ und auf Band 1 hinter dem $\#$ der String $1^a 0$ (wobei \square durch $1^0 = 0$ ersetzt wird), so sucht U in x auf Band 2 einen String $1^p 0 1^a 0 1^q 0 1^b 0 1^d 0$ für ein q , ein b und ein d . Findet es keinen solchen String, so stoppt U und akzeptiert, gdw. $1^p 0$ der Code eines akzeptierenden Zustandes in x ist. Ansonsten ersetzt U den String $1^p 0$ auf Band 3 durch $1^q 0$, den String $1^a 0$ hinter dem $\#$ auf Band 1 durch $1^b 0$ und verschiebt das $\#$ auf Band 1 hinter die nächste 0 rechts, falls $d = 1$, bzw. die nächste 0 links, falls $d = -1$ (wobei \square durch $1^0 = 0$ ersetzt wird). Da die Strings $1^a 0$ und $1^b 0$ nicht gleichlang sein müssen, muß U den Inhalt rechts von dem $\#$ auf Band 2 entsprechend verschieben. (Das kann vermieden werden,

wenn wir auf Band 1 das Zeichen a statt durch $1^a 0$ immer durch $1^a 0^{\gamma-a}$ codieren, wobei γ die Größe des Arbeitsalphabets der Maschine T_x ist, die U aus dem Turingprogramm x entnehmen kann. Dann ist jedes Zeichen durch einen String fester Länge codiert und ein Verschieben der Bandinhalte erübrigt sich.), (2).

Wie wir schon gesehen haben, können wir zu U auch eine äquivalente 1-Band-TM U' finden, die H erkennt.

Das Selbstanwendbarkeitsproblem S ist dann ebenfalls erkennbar, da $x \in S$ gdw. $x \# x \in H$. ■

Definition 3.4.4 (UTM)

Eine Turingmaschine, die H erkennt, nennen wir *universelle Turingmaschine (UTM)*, weil sie jede andere Turingmaschine simulieren kann (man kann sie sich wie ein Betriebssystem vorstellen, das jedes eingegebene Programm ausführt). Entsprechend wird H auch manchmal die *universelle Sprache* genannt. Ebenso wird S manchmal die *Diagonalsprache* genannt, weil ihre Worte x Elementen (x, x) auf der Diagonalen von $\Sigma^* \times \Sigma^*$ entsprechen.

Notation 3.4.5

- \mathbb{A} = Menge der aufzählbaren Sprachen (über Σ),
- \mathbb{E} = Menge der entscheidbaren Sprachen,
- \mathbb{A}^{co} = Menge der co-aufzählbaren Sprachen (Sprachen mit aufzählbarem Komplement),
- \mathbb{E}^{co} = Menge der co-entscheidbaren Sprachen (Sprachen mit entscheidbarem Komplement).

Wir wissen: $\mathbb{E} = \mathbb{E}^{co}$ und $\mathbb{E} = \mathbb{A} \cap \mathbb{A}^{co}$. Aber ist $\mathbb{E} \neq \mathbb{A}$ und $\mathbb{A} \neq \mathbb{A}^{co}$ oder sind sie gleich? Wir wissen ebenfalls, daß S wie H aus \mathbb{A} . Aber es ist $\bar{S} \notin \mathbb{A}$:

Satz 3.4.6 ($\bar{S} \notin \mathbb{A}$)

Das Komplement des Selbstanwendbarkeitsproblems ist nicht aufzählbar.

Beweis: Wäre \bar{S} erkennbar, so gäbe es eine TM T_x , die genau auf den Worten aus \bar{S} stoppt. Damit ergäbe sich folgender Widerspruch:

$$\begin{array}{ccccc}
 T_x(x) \downarrow & \text{gdw.} & x \in \bar{S} & \text{gdw.} & T_x(x) \uparrow \\
 & | & & | & \\
 & T_x \text{ erkennt } \bar{S} & & \text{Def.von } \bar{S} &
 \end{array}$$

Anmerkung 3.4.7

Anfang des 20. Jahrhunderts wurde die von Cantor begründete naive Mengenlehre in eine Krise gestürzt durch ein ganz einfaches Argument von dem Mathematiker Russel. In der Cantorschen Mengenlehre war nicht ausgeschlossen, das unter den Elementen einer Menge diese Menge selbst vorkommt, daß also eine Menge sich selbst enthält. Russel zeigte nun, daß es keine Menge gibt, die alle Mengen enthält, die sich selbst nicht enthalten. Er benützte dabei genau das obige Argument: gäbe es eine Menge, die alle Mengen enthält, die sich selbst nicht enthalten, so würde sie sich selbst enthalten gdw. sie sich selbst nicht enthielte. Es ist dasselbe Argument, das man populärer als das Paradox von dem Barbier kennt, der alle Leute im Dorf rasiert, die sich nicht selbst rasieren. Dieser Barbier weiß nicht, ob er sich nun selber rasieren soll oder nicht. Sie können auch in eine Bibliothek gehen und versuchen einen Katalog aller der Bücher zu machen, die sich nicht selbst referieren, sich nicht selbst aufführen. Es wird Ihnen nicht gelingen, da Sie am Ende entscheiden müßten, ob der Katalog sich selbst nun aufführen soll oder nicht.

Korollar 3.4.8 ($S \notin \mathbb{E}$)

Das Selbstanwendbarkeitsproblem ist nicht entscheidbar.

Beweis: Sonst wäre $\bar{S} \in \mathbb{E} \subseteq \mathbb{A}$. ■

Anmerkung 3.4.9

Damit haben wir die obigen Fragen beantwortet: $\mathbb{A} \neq \mathbb{E} \neq \mathbb{A}^{co} \neq \mathbb{A}$

Korollar 3.4.10 ($H \notin \mathbb{E}$, $H \in \mathbb{A}$, $\bar{H} \in \mathbb{A}^{co}$)

Das Halteproblem ist aufzählbar, aber nicht entscheidbar. Sein Komplement ist co-aufzählbar.

Beweis: Könnte man H entscheiden, so auch S . ■

Noch ein weiteres Beispiel:

Definition 3.4.11

$L_T = \{x \mid \forall y : T_x(y) \downarrow\}$ ist die Menge der Worte, die eine Turingmaschine beschreiben, die immer - auf jedem Input - stoppt (salopp: die Menge der totalen - immer haltenden - Turingmaschinen).

Satz 3.4.12 ($L_T \notin \mathbb{A}$)

Die Menge der immer haltenden Turingmaschinen ist nicht aufzählbar.

Beweis: Beweis durch Diagonalisierung:

Sei L_T aufzählbar, vermöge der berechenbaren Bijektion $a : \mathbb{N} \longleftrightarrow L_T$. Sei $b : \mathbb{N} \longleftrightarrow \Sigma^*$ eine bijektive Aufzählung von Σ^* . Definiere M vermöge: Ist der Input von M das i -te Wort aus Σ^* , so simuliert M das i -te Turingprogramm aus L_T , verändert aber am Schluß der Simulation den Bandinhalt, d.h. für jedes i gilt: M verhält sich anders als das i -te Turingprogramm aus L_T , wenn man ihm das i -te Wort aus Σ^* als Input gibt. M stoppt auf jedem Input, d.h. ein Turingprogramm für M müßte selbst in L_T liegen, was aber nicht sein kann, wenn sich M anders verhält als jedes Turingprogramm aus L_T .

Genauer:

- (1) M ersetzt Input y durch $a(b^{-1}(y)) \# y$, (2).
- (2) M wendet die UTM U an, (3)
- (3) M läuft zum nächsten \square und druckt eine 1 (M verändert den Bandinhalt).

TM M stoppt bei jedem Input. Ist x ein Turingprogramm von M (also $M = T_x$), so ist $x \in L_T$. Damit gibt es ein i mit $a(i) = x$. Per Konstruktion verhält sich aber $M(b(i))$ anders als $T_{a(i)}(b(i))$, da $M(b(i))$ nach der Simulation von $T_{a(i)}(b(i))$ durch die UTM den Bandinhalt ändert, damit kann x nicht das Turingprogramm $a(i)$ sein. ■

3.5 Reduktion

Wir wollen den letzten Satz nochmal anders beweisen, um vorzustellen, wie man ein Problem auf ein bekanntes zurückführt, und diesen Prozeß formalisieren zu dem Instrument der Reduktion, das diese Rückführung so abstrahiert, daß man seine Überlegungen auf wenige Bedingungen konzentrieren kann. Zunächst die schwächere Version des Satzes 3.4.12:

Satz 3.5.1 ($L_T \notin \mathbb{E}$)

Die Menge der immer haltenden Turingmaschinen ist nicht entscheidbar.

Beweis: Wir wissen, daß $S \notin \mathbb{E}$. Wir zeigen: wäre L_T entscheidbar, so auch S . Um diesen Widerspruch herbeizuführen, nehmen wir an, es gäbe eine TM M_T , die das Problem L_T entscheidet und wir

suchten eine, die das Problem S löst, indem wir das "ungelöste" Problem S auf das schon "bekannte", d.h. gelöste Problem L_T zurückführen.

Unsere Frage ist, ob $x \in S$. Diese Frage führen wir zurück auf eine Frage der Art $y \in L_T$, wir wollen also die Frage $x \in S?$ umbauen in eine Frage $y \in L_T?$, so daß die Antwort auf die Frage nach y uns auch eine Antwort auf die Frage nach x liefert.

Wenn es uns gelingt, eine Turingmaschine M_f anzugeben, die x so in y umbaut, daß x in S liegt gdw. y in L_T liegt, und wenn es tatsächlich eine TM M_T gibt, die das Problem L_T entscheidet, so hätten wir auch eine TM, die S entscheidet, nämlich die Serienschaltung $M_T \circ M_f$ (M_T nach M_f): erst wird x in y verwandelt, dann wird festgestellt, ob $y \in L_T$. Da wir allerdings wissen, daß S nicht entscheidbar ist, muß also unsere Annahme falsch gewesen sein, d.h. L_T kann nicht entscheidbar sein.

Dieser Umbau von x zu y ist freilich leicht, denn wir müssen nur der TM T_x eine feste TM E_x vorschalten, die den Input durch x ersetzt. Sei $x = a_1 \cdots a_n$. Dann sind die Befehle von E_x :

$$\begin{aligned} & \{(0, a, 0, \square, 1) \mid a \in \Sigma\} \cup (\text{löscht den Input}) \\ & \{(0, \square, 1, a_1, 1), (1, \square, 2, a_2, 1), \dots, (n-1, \square, n, \square, -1)\} \cup (\text{druckt } x) \\ & \{(n, a, n, a, -1)(n, \square, n+1, \square, 1) \mid a \in \Sigma\} \text{ (läuft zum Anfang von } x) \end{aligned}$$

Nun sei y das Turingprogramm der Serienschaltung $T_x \circ E_x$. Es ist leicht einzusehen, daß es eine Turingmaschine gibt, die jedes x auf diese Weise in y umformt, daß also die Funktion $f : \Sigma^* \rightarrow \Sigma^*$ mit $f(x) = y$ durch eine Turingmaschine M_f berechnet werden kann.

Sei e das Turingwort von E_x . Wandle e in e' und x in x' gemäß ■

Jetzt wollen wir Satz 3.4.12 so beweisen:

Satz 3.5.2 ($L_T \notin \mathbb{A}$)

Die Menge der immer haltenden Turingmaschinen ist nicht aufzählbar.

Beweis: Wir wissen $\bar{S} \notin \mathbb{A}$. Wir nehmen an, es sei $L_T \in \mathbb{A}$ und M_T die Turingmaschine, die L_T erkennt. Jetzt wollen wir das Problem \bar{S} mithilfe dieses "bekannten" Problems lösen, indem wir aus einer Frage an \bar{S} eine an L_T machen. Wenn uns das gelingt, so können wir \bar{S} mithilfe von L_T aufzählen, was unserem Wissen widerspricht. Also muß unsere Annahme falsch sein.

Dies gelingt uns auf folgende Weise: wir wandeln das TP x so in ein TP $y = f(x)$ um, daß $x \in \bar{S}$ gdw. $y \in L_T$, d.h. eine Antwort auf y ist auch eine auf x . Die TM $T_y(w)$ arbeite wie folgt:

Simuliere $T_x(x)$ $|w|$ Schritte lang. Stoppt $T_x(x)$ in dieser Zeit, so zykle, sonst stoppe. Stoppt $T_x(x)$ nicht, stoppt T_y offenbar auf allen Inputworten (jeder Länge), stoppt aber $T_x(x)$, so wird T_y auf allen Inputs genügender Länge zykeln.

Wir beschreiben T_y etwas genauer: zunächst definieren wir eine 2-Band-TM:

```
%(TM DRUCKx:)
  Drucke  $x$  auf Band 2;
%(TM SIM:)
  while Kopf 1 liest nicht  $\square$ , do begin
    if  $T_x$  auf Band 2 stoppt then zykle
    else begin  $T_x$  macht einen Schritt auf Band 2;
              Kopf 1 rückt ein Feld nach rechts
    end
  end{while}
```

Noch genauer: Die TM SIM hat folgende Befehle:

$\{(p, c, a, q, c, b, 1, d) \mid c \in \Sigma\}$ falls $(p, a, q, b, d) \in \delta_x$
 (simuliert T_x solange auf Band 1 nicht \square gelesen wird bzw. stoppt wenn w überlaufen worden ist)

$\{(p, c, a, p, c, a, 1, 0) \mid c \in \Sigma \cup \{\square\}\}$ falls kein Befehl von δ_x mit p, a beginnt (geht in den Zykelzustand z , da T_x stoppt).

Die TM DRUCK x ist eine Variante von Ex oben. Als Aufgabe sei hier gegeben, zu zeigen, daß es eine TM M_{Bd} gibt, die jedes 2-Band-TP x in ein äquivalentes 1-Band-TM x' umwandelt: $M_{Bd}(x) \downarrow x'$. Also gelingt es eine TM M_f anzugeben, die aus x das TP y baut: $M_f(x) \downarrow y$.

Wäre also L_T erkennbar durch M_T , so wäre auch \bar{S} erkennbar durch $M_T \circ M_f$, was unserem Wissen widerspricht. ■

Noch ein Versuch, diesmal kein Widerspruchsbeweis:

Definition 3.5.3

$S_\varepsilon = \{w \mid T_w(w) \downarrow \varepsilon\}$ ist die Menge der Turingworte, die angewandt auf sich selbst mit leerem Band stoppen.

Satz 3.5.4 ($S_\varepsilon \in \mathbb{A}$)

Die Menge der Turingmaschinen, die auf sich selbst angewandt mit leerem Band stoppen, ist aufzählbar.

Beweis:

1. Versuch [direkte Aufzählung]: Sei $w : N \rightarrow \Sigma^*$ eine Aufzählung von Σ^* . Wir konstruieren eine TM M die S_ε aufzählt: $M(l)$ simuliert $T_{w(i)}(w(i))$ für alle $i = 0, 1, 2, \dots$ bis l Simulationen mit Output ε gestoppt haben. Dann gibt sie $w(i)$ aus. Da aber viele Turingmaschinen auf ihrem eigenen Turingprogramm nicht stoppen, würde M bei diesem Verfahren bei der ersten TM dieser Art hängen bleiben. Also simuliert M jede TM nur eine gewisse Anzahl an Schritten und nimmt dann eine andere TM dran, um die alte später eine größere Zahl an Schritten noch einmal zu simulieren. Genauer, sie simuliert die erste TM einen Schritt, dann die ersten beiden TMen zwei Schritte, etc, allgemein die ersten i TMen i Schritte. So wird jede TM mit einer immer größeren Zahl an Schritten simuliert, bis l TMen erfolgreich (bis zum Stop mit Output ε) simuliert werden konnten. Das Wort, auf dem die letzte Turingmaschinensimulation erfolgreich war, wird dann als Output ausgegeben.

```

j = 0, k = 0, S := ∅
(j = Zahl der bisher erfolgreichen Simulationen,
 S = Menge der TPe, die bisher mit ε gestoppt haben)
while k < ∞ do begin
  for i = 1 to k do begin {die ersten k TMen werden k Schritte simuliert}
    if Tw(i)(w(i)) innerhalb von k Schritten mit Output ε stoppt and i ∉ S
      then if j = l then Return(w(i)) else begin iarrow S; j := j + 1 end;
    end{for};
  k := k + 1;
end{while}

```

2. Versuch[Rekursion]: Wir führen das uns noch unbekannt Problem S_ε zurück auf das bekannte Problem S , genauer, wir führen die für uns ungelöste Frage, ob $x \in S_\varepsilon$, zurück auf eine Frage der Form $y \in S$, für die wir bereits einen Lösungsalgorithmus, d.h. eine TM M_S , haben. (Lösung heißt hier allerdings nur einen Erkennungsalgorithmus zu haben: die TM beantwortet unsere Frage mit ja, wenn $y \in S$, aber u.U. gar nicht, falls $y \notin S$.)

Wir wollen also die Frage " $x \in S_\varepsilon$?" umbauen in eine Frage " $y \in S$ ", so daß die Antwort auf die Frage nach y uns auch eine Antwort auf die Frage nach x liefert. Wenn es uns gelingt, eine Turingmaschine M_f anzugeben, die x so in y umbaut, daß y genau dann in S liegt, wenn x in S_ε liegt, dann erkennt die Serienschaltung $M_S \circ M_f$ die Sprache S_ε , d.h. $S_\varepsilon \in \mathbb{A}$.

Wie sieht das Turingprogramm y aus? Es soll ja gelten: $x \in S_\varepsilon \iff y \in S$, d.h. $T_x(x) \downarrow \varepsilon \iff T_y(y) \downarrow$. Es ist einfacher, wenn wir y so bauen, daß T_y für alle Inputs w stoppt, wenn $T_x(x) \downarrow \varepsilon$, andernfalls aber für keinen Input, dann ist ja insbesondere die obige Forderung erfüllt.

Also arbeite T_y wie folgt: es ersetzt den Input durch x , wendet dann die TM T_x an. Stoppt $T_x(x)$, so überprüft T_y , ob das Band leer ist. Wenn ja, so stoppt auch T_y , wenn nein, so zyckelt T_y .

Genauer: Betrachten wir die TM T_x . Ersetzen wir jeden Befehl (p, a, q, \square, d) mit $a \in \Gamma_o$ durch $(p, a, q, \blacksquare, d)$ und fügen zu jedem Befehl (p, \square, q, a, d) mit $a \in \Gamma$ noch $(p, \blacksquare, q, a, d)$ hinzu, so bekommen wir eine äquivalente TM $T_{x'}$, die aber nie \square druckt. Die obige TM E_x ersetzt den Input durch x . Sei CHECK die TM, die den Bandinhalt am Schluß auf Leerheit überprüft:

$\{(1, \blacksquare, 1, \blacksquare, -1), (1, x, z, x, 0), (1, \square, 2, \square, 1) \mid x \in \Gamma\} \cup$
 (läuft zum linken Randblank, geht in Zykelzustand z , wenn ein Symbol $\neq \blacksquare$ gelesen wird)
 $\{(2, \blacksquare, 2, \blacksquare, 1), (2, x, z, x, 0), (2, \square, h, \square, 0) \mid x \in \Gamma\} \cup$
 (läuft zum rechten Randblank, geht in Zykelzustand z , wenn ein Symbol $\neq \blacksquare$ gelesen wird)
 $\{(z, x, z, x, 1) \mid x \in \Gamma_o\}$
 (zyckelt).

Jetzt schalten wir die Maschinen E_x , $T_{x'}$ und CHECK in Serie und codieren das Ergebnis als ein Turingprogramm. Das sei $f(x)$. Es ist nun eine Routineaufgabe, eine Turingmaschine M_f zu konstruieren, die aus x dieses $f(x)$ herstellt. ■

Was haben wir gemacht? Für die Frage " $y \in S$ " haben wir eine Antwort. Die Idee war, das Problem x so in ein Problem $f(x)$ umzuformen, daß eine Antwort auf die Frage " $f(x) \in S$ ", auch schon die Frage nach " $x \in S_\varepsilon$ " beantwortete. Sollen die Antworten berechenbar sein, so muß auch die Umformung berechenbar sein, d.h. von einer TM geleistet werden können.

Damit haben wir: haben wir eine TM, die B entscheidet oder erkennt, und gelingt es uns eine berechenbare Umformungsabbildung $f : \Sigma^* \rightarrow \Sigma^*$ zu finden, sodaß $x \in A$ gdw. $f(x) \in B$, so gibt es auch eine TM, die A entscheidet oder erkennt.

Jetzt wollen wir diese Rückführungen eines unbekanntes Problems auf ein bekanntes formalisieren:

Definition 3.5.5 (many-one-Reduzierbarkeit)

A ist *many-one-reduzierbar* auf B (in Zeichen: $A \leq_m B$) gdw. es gibt eine totale berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ mit

- $w \in A$ gdw. $f(w) \in B$
- oder äquivalent $A = f^{-1}(B)$
- oder äquivalent $f(A) \subseteq B$ und $f(\bar{A}) \subseteq \bar{B}$.

Der Name „many-one“ bezieht sich auf die Eigenschaft der Reduktionsfunktion, die viele Worte auf eines abbilden kann, im Gegensatz zu one-one-Reduktionen, bei denen die reduzierende Funktion eineindeutig ist. Daneben gibt es noch eine Reihe anderer Reduktionsbegriffe wie Turingreduzierbarkeit oder Truth-table-Reduzierbarkeit, die wir hier aber nicht behandeln.

In dieser Notation würden wir die beiden vorigen Beweise wie folgt schreiben:

1. $S_\varepsilon \notin \mathbb{E} : S \leq_m S_\varepsilon$ vermöge der Reduktion $v : x \mapsto v(x)$, wobei $T_{v(x)}$ den Input durch x ersetzt, dann die TM T_x anwendet und dann das Band löscht.
2. $S_\varepsilon \in \mathbb{A} : S_\varepsilon \leq_m S$ vermöge der Reduktion $u : x \mapsto u(x)$, wobei $T_{u(x)}$ den Input durch x ersetzt, dann die TM T_x anwendet und genau dann stoppt, wenn $T_x(x) \downarrow \varepsilon$.

Man muß bei dieser kurzen Beweisskizze allerdings voraussetzen, daß die Berechenbarkeit der Reduktion (Reduktionsabbildung) klar ist.

Aufgabe 3.5.6

$A \leq_m B \implies$ 1. $B \in \mathbb{E} \implies A \in \mathbb{E}$
 2. $B \in \mathbb{A} \implies A \in \mathbb{A}$.

Anmerkung 3.5.7

Diese Aussage berechtigt uns zu der durch das Zeichen \leq_m suggerierten Vorstellung, daß $A \leq_m B$ bedeutet, daß A ein leichteres oder einfacheres Problem ist als B : ist B erkennbar, so auch A , ist B gar entscheidbar, so auch A .

Aufgabe 3.5.8

$A \leq_m B \implies \bar{A} \leq_m \bar{B}$.

Satz 3.5.9

$A \in \mathbb{A} - \mathbb{E} \implies A \not\leq_m \bar{A}$ und $\bar{A} \not\leq_m A$.

Beweis: $\bar{A} \leq_m A \implies A \in E$ (wegen Satz 3.2.18) ■
 $A \leq_m \bar{A} \iff \bar{A} \leq_m A$ (wegen Aufgabe 3.5.8)

Satz 3.5.10

$A \leq_m H \iff A \in \mathbb{A}$.

Beweis: \implies : $H \in \mathbb{A}$ (Satz 3.4.3), somit auch $A \in \mathbb{A}$ (Aufgabe 3.5.6). ■
 \impliedby : Sei M_x eine TM, die A erkennt. Dann ist $A \leq_m H$ vermöge $f : w \rightarrow x\#w$.

Definition 3.5.11 (hart, vollständig)

Sei \mathbb{K} eine Klasse von Sprachen aus Σ^* und $A \subseteq \Sigma^*$.

A heißt *many-one hart* (engl. hard) in \mathbb{K} gdw $L \in \mathbb{K} \implies L \leq_m A$ (alle Sprachen aus \mathbb{K} lassen sich auf A reduzieren).

A heißt *many-one vollständig* (engl. complete) in \mathbb{K} gdw. A many-one hart und $A \in \mathbb{K}$.

Satz 3.5.12

H ist (many-one) vollständig in \mathbb{A} .

Beweis: $H \in \mathbb{A}$ (Satz 3.4.3). ■
 $L \in \mathbb{A} \implies L \leq_m H$ (Satz 3.5.10).

Aufgabe 3.5.13

A vollständig in \mathbb{A} , $A \leq_m B$, $B \in \mathbb{A} \implies B$ vollständig in \mathbb{A}

Definition 3.5.14 (many-one äquivalent)

A *many-one äquivalent* zu B (i.Z. $A \equiv_m B$) gdw. $A \leq_m B$ und $B \leq_m A$.

Satz 3.5.15

$S \equiv_m H$.

Beweis: $S \leq_m H$ vermöge der Reduktion $f : x \rightarrow x\#x$.

$H \leq_m S$ vermöge der Reduktion $g : x\#y \rightarrow z$, wobei die Turingmaschine T_z den Input durch y ersetzt und dann T_x anwendet (d.h. $z = e_y \circ x$, wo e_y das Turingprogramm der Turingmaschine, die den Input durch y ersetzt). T_z stoppt für alle Inputs, also auch auf seinem eigenen Turingprogramm z , wenn $T_x(y)$ stoppt. Stoppt aber $T_x(y)$ nicht, so stoppt T_z für keinen Input, also auch nicht auf z . ■

3.5.1 Weitere Sprachen

Satz 3.5.16

$L_\varepsilon = \{x \mid \forall w : T_x(w) \downarrow \varepsilon\} \notin \mathbb{E}$.

Beweis: $S \leq_m L_\varepsilon$ vermöge $f : x \rightarrow z$, wobei T_z den Input durch x ersetzt, dann T_x anwendet und schließlich das Band löscht (Also $z = e_x \circ x \circ l$, wobei e_x wie oben und l das Turingprogramm der TM, die das ganze Band löscht und dann stoppt). ■

Satz 3.5.17

$\bar{L}_\emptyset := \{x \mid \exists w : T_x(w) \downarrow\} \notin \mathbb{E}$.

Beweis: $S \leq_m \bar{L}_\emptyset$ vermöge f aus vorigem Beweis. ■

Satz 3.5.18

Folgende Sprachen sind nicht aus \mathbb{E} :

$$\begin{aligned} H_1 &= \{x \mid T_x(\varepsilon) \downarrow\} \\ L_t &= \{x \mid \forall w : T_x(w) \downarrow\} \\ L_\infty &= \{x \mid \exists^\infty w : T_x(w) \downarrow\}. \end{aligned}$$

Beweis: Sei L eine dieser Sprachen. Dann gilt $S \leq_m L$ vermöge $h : x \rightarrow z$, wo T_z den Input durch $x\#x$ ersetzt und dann die universelle TM U für H anwendet. ■

Satz 3.5.19

$H_3 = \{x\#y\#z \mid T_x(y) \downarrow z\} \notin \mathbb{E}$. H_3 ist vollständig in \mathbb{A} .

Beweis: $S \leq_m H_3$ vermöge $g : x \rightarrow f(x)\#x\#\varepsilon$ ($f(x)$ wie in Beweis von Satz 3.5.16). ■

Satz 3.5.20

$\bar{L}_\emptyset := \{x \mid \exists w : T_x(w) \downarrow\} \in \mathbb{A}$.

Beweis:

1. (Direkte Konstruktion) Sei $w : \{1\}^* \rightarrow \Sigma^*$ eine Aufzählung von Σ^* . Wir konstruieren ein TM M , die \bar{L}_\emptyset erkennt: $M(x)$, wendet die TM T_x auf alle Inputs $w(1^i)$ für $i = 0, 1, 2, \dots$ an bis T_x auf einem davon stoppt. Dies muß M wieder im dove-tailing machen, d.h. sie simuliert T_x auf den ersten i Worten i Schritte lang für $i = 1, 2, \dots$

2. (Beweis durch Reduktion) $\bar{L}_\emptyset \leq_m S$ vermöge f wobei $T_{f(x)}$ gerade die TM M aus 1 ist. ■

Aufgabe 3.5.21

$H_1, H_3 \in \mathbb{A}$.

Satz 3.5.22

$L_\varepsilon := \{x \mid \forall w : T_x(w) \downarrow \varepsilon\} \notin \mathbb{A}$.

Beweis:

1. Beweis per Reduktion: $\bar{S} \leq_m L_\varepsilon$ vermöge h , wobei $T_{h(x)}(t) \downarrow \varepsilon$ falls $T_x(x)$ nicht innerhalb t Schritten stoppt. Andernfalls zyckelt $T_{h(x)}(t)$.

2. Beweis per Diagonalisierung: Könnte man L_ε aufzählen, gäbe es also eine Aufzählung $a : \mathbb{N} \rightarrow L_\varepsilon$, so könnte man eine TM T_x konstruieren, die auf Input $w(i)$ die TM $T_{a(i)}$ (auf dem i -ten Wort, die i -te TM der Aufzählung) anwendet, dann aber noch einen Schritt mehr tut. Da T_x auf jedem Input mit ε stoppt, muß $x \in L_\varepsilon$ sein. Wäre also $x = a(i)$ für ein i , so müßte $T_x(w(i))$ genausoviele Schritte tun

wie $T_{a(i)}(w(i))$ im Widerspruch zur Konstruktion von T_x . ■

Satz 3.5.23

$\overline{L}_T, L_\emptyset, L_T, L_\infty \notin \mathbb{A}$.

Beweis: $\overline{S} \leq \overline{L}_T, L_\emptyset$ vermöge f aus dem Beweis von Satz 3.5.16. $\overline{S} \leq L_T, L_\infty$ vermöge h Beweis von Satz 3.5.22. ■

Eine interessante Sprache ist die Sprache der äquivalenten Turingprogramme. Der nächste Satz sagt, daß diese Sprache nicht nur nicht entscheidbar ist, sondern daß sie nicht einmal aufzählbar oder co-aufzählbar ist. Allgemein bedeutet das, daß es kein automatisches Verfahren gibt, daß für ein Programm entscheidet, ob es das tut, was in einer anderen universellen Sprache (Semantik) beschrieben ist. Damit wissen wir aber, daß eine uneingeschränkte Programmverifikation nur heuristisch möglich ist. So versteht man auch, warum diese Disziplin auf so große Schwierigkeiten stößt.

Satz 3.5.24

$L_\sim := \{x\#y \mid T_x \equiv T_y\} = \{(x, y) \mid \forall w : T_x(w) \downarrow z \iff T_y(w) \downarrow z\} \notin \mathbb{A} \cup \mathbb{A}^{co}$.

Beweis:

1. Wir zeigen $L_\sim \notin \mathbb{A}^{co}$, indem wir zeigen, daß die nicht co-aufzählbare Sprache S "leichter" ist. $S \leq L_\sim$ vermöge $g : x \rightarrow z\#l$, wobei z aus Beweis von Satz 3.5.16 und l ein TP für die TM, die das Band löscht und stoppt, die also auf jedem Input mit ε stoppt.

2. Wir zeigen $L_\sim \notin \mathbb{A}$, indem wir zeigen, daß die nicht aufzählbare Sprache \overline{S} "leichter" ist. $\overline{S} \leq_m L_\sim$ vermöge $h : x \rightarrow z\#u$, wobei z aus Beweis von Satz 3.5.16 ($T_z(-) \downarrow \varepsilon \iff T_x(x) \downarrow$) und u ein TP für die TM, die auf jedem Input unendlich läuft (zykelt ($\forall w : T_u(w) \uparrow$)). ■

Wir halten fest:

- \mathbb{A} -vollständig sind: $S, S_\varepsilon, H, H_1, H_3, \overline{L}_\emptyset$.
- nicht aus \mathbb{A} sind: $\overline{S}, \overline{S}_\varepsilon, \overline{H}, \overline{H}_1, \overline{H}_3, L_\emptyset, \overline{L}_\varepsilon, \overline{L}_t$.
- nicht aus $\mathbb{A} \cup \mathbb{A}^{co}$ sind: $L_\varepsilon, L_\infty, L_t, L_\sim$.

3.6 Die Sätze von Rice

Notation 3.6.1

Sei $\mathbb{P} = \{f : \Sigma^* \rightarrow \Sigma^* \mid f \text{ 1-stellige partiell berechenbare Funktion}\}$ die Menge der (1-stelligen) partiellen berechenbaren Funktionen $\Sigma^* \rightarrow \Sigma^*$. Sei φ_x die von TM T_x (mit dem TP x) berechnete Funktion.

Definition 3.6.2

$\mathfrak{B} \subseteq \mathbb{P}$ heißt *stark entscheidbar (aufzählbar)* gdw. $B = \{x \mid \varphi_x \in \mathfrak{B}\}$ entscheidbar (aufzählbar).

$\mathfrak{B} \subseteq \mathbb{P}$ heißt *schwach entscheidbar (aufzählbar)* gdw. es eine entscheidbare (aufzählbare) Menge $B \subseteq \Sigma^*$ gibt mit $\mathfrak{B} = \{\varphi_x \mid x \in B\}$.

Eine Funktionenmenge \mathfrak{B} ist also stark entscheidbar, wenn die Menge B aller Turingmaschinen, die Funktionen aus \mathfrak{B} berechnen, entscheidbar ist. Sie ist schwach entscheidbar, wenn es eine entscheidbare Menge von Turingmaschinen gibt, die ausreichen, um alle Funktionen aus \mathfrak{B} zu berechnen. Der erste Satz von Rice sagt nun, daß keine Funktionenmenge (außer der leeren und der vollen) stark entscheidbar ist. Das bedeutet z.B., daß es nicht entscheidbar ist, ob eine Turingmaschine eine

vorgegebene Funktion berechnet. Das bedeutet: es gibt kein allgemeines Verfahren, das Ihnen sagt, ob Ihr Programm, das Sie für eine gegebene Aufgabe geschrieben haben, diese Aufgabe auch wirklich löst.

Satz 3.6.3 (Rice 1)

Sei \mathfrak{B} eine Funktionenmenge aus \mathbb{P} mit $\mathfrak{B} \neq \emptyset$ und $\mathfrak{B} \neq \mathbb{P}$ (\mathfrak{B} ist nicht trivial). Dann ist \mathfrak{B} stark unentscheidbar (nicht stark entscheidbar). (Nur triviale Funktionsmengen sind stark entscheidbar.)

Beweis: 1. Sei die leere (nirgends definierte) Funktion \emptyset nicht in \mathfrak{B} und g eine Funktion in \mathfrak{B} ($\neq \emptyset$) und M eine TM für g . $S \leq_m B = \{x \mid \varphi_x \in \mathfrak{B}\}$ vermöge $f : x \rightarrow y$, wobei y das TP der TM T_y ist, die auf Band 2 $T_x(x)$ simuliert, dann Band 2 löscht und dann auf Band 1 die TM M anwendet. D.h. T_y berechnet g , wenn $x \in S$, und sonst die Funktion \emptyset .

2. Sei die leere (nirgends definierte) Funktion \emptyset in \mathfrak{B} , g eine Funktion nicht in \mathfrak{B} ($\neq \mathbb{P}$) und M eine TM für g . $\bar{S} \leq_m B = \{x \mid \varphi_x \in \mathfrak{B}\}$ vermöge $f : x \rightarrow y$, wobei y das TP der TM T_y ist, die auf Band 2 zunächst $T_x(x)$ simuliert und dann auf Band 1 die TM M anwendet. D.h. T_y berechnet \emptyset , wenn $x \notin S$, und sonst die Funktion g . ■

Korollar 3.6.4

Die folgenden Sprachen sind nicht entscheidbar:

$$\begin{array}{ll}
 L_\varepsilon & = \{x \mid \forall y : T_x(y) \downarrow \varepsilon\} & \mathfrak{B}_\varepsilon = \{\text{die totale konstante Funktion } \varepsilon\} \\
 \bar{L}_\emptyset & = \{x \mid \exists y : T_x(y) \downarrow\} & \bar{\mathfrak{B}}_\emptyset = \text{Menge der nichtleeren Funktionen,} \\
 L_T & = \{x \mid \forall y : T_x(y) \downarrow\} & \mathfrak{B}_T = \text{Menge der totalen Funktionen,} \\
 \bar{L}_T & = \{x \mid \exists y : T_x(y) \uparrow\} & \bar{\mathfrak{B}}_T = \text{Menge der nichttotalen Funktionen,} \\
 L_\emptyset & = \{x \mid \forall y : T_x(y) \uparrow\} & \mathfrak{B}_\emptyset = \{\text{die leere Funktion}\}, \\
 L_\infty & = \{x \mid \overset{\infty}{\exists} y : T_x(y) \downarrow\} & \mathfrak{B}_\infty = \text{Menge der unendlich oft definierten Funktionen.}
 \end{array}$$

Definition 3.6.5 (code f)

Sei f eine endliche Funktion aus \mathbb{P} (nur auf endlich vielen Worten definiert). Eine solche Funktion können wir in ein Wort codieren: $\text{code } f = x_1 \# y_1 \# \dots \# x_n \# y_n$ wo $\{x_1, \dots, x_n\}$ der Definitionsbereich von f ist, $y_i = f(x_i)$ und $x_i < x_j$ falls $i < j$ ($i = 1, \dots, n$). Wir schreiben also die Paare der Funktion hintereinander, geordnet nach der ersten Komponente.

Definition 3.6.6 (Fortsetzung einer Funktion)

Seien $f, g \in \mathbb{P}$. g heißt Fortsetzung von f gdw.

1. $\text{Def } f \subseteq \text{Def } g$
2. $\forall x \in \text{Def } f : f(x) = g(x)$

Der zweite Satz von Rice sagt, daß \mathfrak{B} stark aufzählbar ist, nur wenn \mathfrak{B} auf eine etwas andere Weise trivial ist, wenn nämlich \mathfrak{B} gerade die Menge aller Fortsetzungen einer "aufzählbaren" Menge endlicher Funktionen ist. Immer wenn dies nicht der Fall ist, etwa wenn \mathfrak{B} nur eine Funktion enthält, ist \mathfrak{B} nicht stark aufzählbar. Damit ist das Überprüfen der semantischen Korrektheit eines Programms in noch weitere Ferne gerückt.

Zuvor aber noch folgende technische Definition:

Definition 3.6.7

Seien M_1 und M_2 1-Band-TMen. $M = [M_1 \parallel M_2]$ sei eine 2 Band-TM, die zuerst den Input auf das zweite Band kopiert und dann M_1 auf Band 1 und M_2 auf Band 2 anwendet, wobei sie abwechselnd M_1 einen Schritt und dann M_2 einen Schritt tun läßt. M stoppt sofort, wenn eine der beiden Maschinen stoppt und gibt den Output der stoppenden Maschine als eigenen Output aus.

Satz 3.6.8 (Rice 2)

Sei \mathfrak{B} eine Funktionenmenge aus \mathbb{P} . \mathfrak{B} ist stark aufzählbar gdw. es eine aufzählbare Menge $C \subseteq (\Sigma \cup \{\#\})^*$ gibt mit $\mathfrak{B} = \{g \in \mathbb{P} \mid g \text{ ist Fortsetzung einer endlichen Funktion } f \text{ mit code } f \in C\}$.

Beweis:

I. \implies : Sei $B = \{x \mid \varphi_x \in \mathfrak{B}\}$ erkennbar durch M_B .

Sei \mathfrak{B}_e die Menge der endlichen Funktionen von \mathfrak{B} .

1. Wir zeigen: Dann ist $\text{code}(\mathfrak{B}_e) = \{\text{code } f \mid f \in \mathfrak{B}_e\}$ aufzählbar.

Folgende TM M erkennt $\text{code}(\mathfrak{B}_e)$.

- M überprüft, ob ihr Input w Code einer endlichen Funktion sein kann. Wenn nicht, dann verwirft M .
- Wenn ja, so konstruiert M ein TP $p(w)$, das die vom Code w beschriebene Funktion f_w berechnet (die TM $T_{p(w)}$ hat w im Zustand einprogrammiert. Bei Input x prüft sie, ob es ein Paar $x\#y$ in w gibt. Wenn ja, so ersetzt sie den Input x durch y , wenn nein, so verwirft sie).
- Dann wendet sie auf $p(w)$ die TM M_B an. M akzeptiert gdw. M_B akzeptiert.

2. Wir zeigen: Ist $f \in \mathfrak{B}_e$, so sind alle Fortsetzungen $g \in \mathbb{P}$ von f in \mathfrak{B} :

Annahme: Es gibt eine Fortsetzung g von f , die nicht in \mathfrak{B} ist. Sei weiter M_f eine TM, die f berechnet (o.B.d.A. stoppt $M_f(w)$ gdw. $f(w)$ definiert). Sei M_g eine TM für g und $M_{x,g}$ eine 2-Band-TM, die erst auf Band 2 das Wort x druckt, dann TM T_x auf Band 2 und dann die TM M_g auf Band 1 simuliert. Dann ist $\bar{S} \leq_m B$ vermöge $k : x \rightarrow y$, wo $T_y = [M_f \parallel M_{x,g}]$ die Parallelschaltung der TMen M_f und $M_{x,g}$ ist. Ist $x \in \bar{S}$ so berechnet T_y die Funktion $f \in \mathfrak{B}_e \subseteq \mathfrak{B}$, andernfalls berechnet T_y die Funktion $g \notin \mathfrak{B}$. Das ist ein Widerspruch zu der Voraussetzung, nach der B erkennbar ist.

3. Wir zeigen: Jede Funktion g aus \mathfrak{B} ist Fortsetzung einer endlichen Funktion $f \in \mathfrak{B}_e$. Angenommen, es sei nicht so, es gäbe also eine Funktion g aus \mathfrak{B} , die von keiner endlichen Funktion aus \mathfrak{B}_e Fortsetzung ist. Sei M_g eine TM für g , die nie stoppt bevor sie das Inputende (\square hinter dem Input) erreicht hat, und M_x eine TM, die den Input durch x ersetzt, darauf T_x anwendet und verwerfend stoppt, sobald T_x stoppt.

Dann gilt $\bar{S} \leq_m B$ vermöge $h : x \rightarrow z$, wo $T_z = [M_g \parallel M_x]$ die Parallelschaltung der TMen M_g und M_x ist. Ist $x \in \bar{S}$ so berechnet T_z die Funktion g . Andernfalls verwirft $T_z(w)$ für fast alle w , da $M_g(w)$ immer mindestens $|w|$ Schritte macht. Also berechnet T_z in diesem Fall eine endliche Einschränkung f von g , die nach Annahme nicht aus \mathfrak{B}_e also auch nicht aus \mathfrak{B} ist. Das ist ein Widerspruch zu der Voraussetzung, nach der B erkennbar ist, \bar{S} aber nicht.

II. \impliedby : Sei $C = \{c_1, c_2, c_3, \dots\}$ eine aufzählbare Code-Menge und $B = \{x \mid \varphi_x \text{ ist Fortsetzung einer Funktion } f \text{ mit code } f \in C\}$. Dann erkennt folgende TM M die Menge B : M angewandt auf Input x überprüft, ob die TM T_x eine Fortsetzung einer Funktion mit Code aus C berechnet, ob also T_x für einen Code $c_i = x_{i,1}\#y_{i,1}\#\dots\#x_{i,k_i}\#y_{i,k_i}$ aus C für jeden Input $x_{i,j}$ den Output $y_{i,j}$ ausgibt ($j = 1, \dots, k_i$). Dazu simuliert M die TM T_x im Dove-tailing auf allen ersten Komponenten der Paare von c_1, \dots, c_i je i Schritte ($i = 1, 2, 3, \dots$). Wenn T_x für ein c_i alle Paare erfolgreich überprüft hat, so ist φ_x eine Fortsetzung der von Code c_i beschriebenen endlichen Funktion, d.h. $x \in B$, d.h. M stoppt akzeptierend.

Genauer: Sei $c_i = x_{i,1}\#y_{i,1}\#\dots\#x_{i,k_i}\#y_{i,k_i}$. Dann arbeitet die TM T_x wie folgt:

```

for  $l = 1$  to  $\infty$  do begin
  for  $i = 1$  to  $l$  do begin
     $R = 0^{k_i}$ 
    for  $j = 1$  to  $k_i$  do

```

```

        if  $T_x(x_{i,j}) \downarrow y_{i,j}$  in  $l$  Schritten then  $R[j] = 1$ 
    end;
    if  $R = 1^{k_i}$  then Stop akzeptierend;
end;
end.
    
```

3.7 Die Ackermann-Funktion

Wie wir schon gesehen haben gibt es unter den überabzählbar vielen arithmetischen Funktionen nur abzählbar viele berechenbare. Dennoch fällt es uns nicht leicht, eine davon anzugeben. Das liegt daran, daß wir sehr an das algorithmische Denken gewöhnt sind. Die meisten der uns aus der Schule bekannten Funktionen sind geradezu durch ein Berechnungsverfahren definiert.

Natürlich können wir mit unseren Mitteln bereits leicht die Existenz nicht berechenbarer Funktionen einsehen. Da berechenbare partielle Funktionen einen aufzählbaren Definitionsbereich haben, ist jede Einschränkung einer totalen Funktion auf eine nichtaufzählbare Teilmenge immer eine nichtberechenbare Funktion, z.B. die Einschränkung $\text{id}_{\bar{S}}$ der Identität id auf das Selbstanwendbarkeitsproblem \bar{S} . Allerdings sind dies keine besonders befriedigenden Beispiele. Wie sähe denn eine totale, nicht berechenbare Funktion aus?

Wir wollen eine arithmetische Funktion angeben, die so schnell wächst, daß sie aus diesem Grunde nicht berechenbar sein kann. Dazu wollen wir uns überhaupt erst einmal ein Bild von schnell wachsenden Funktionen machen. Dazu dient dieser Abschnitt, während im darauf folgenden Abschnitt dann eine Funktion vorgestellt wird, die so ungeheuer schnell wächst, daß jedes Verfahren, sie zu berechnen, versagen muß.

3.7.1 Die i -te Ackermannfunktion A_i

Wie können wir schnell wachsende Funktionen notieren? Versuchen wir: 2^n ist eine schnell wachsende Funktion, aber natürlich wächst die Funktion 2^{2^n} bedeutend schneller, nur ist sie ungünstig zu notieren. Kürzen wir sie ab durch ${}^n 2$. Sofort können wir eine schneller wachsende

Funktion notieren: ${}^n 2$. Diese können wir wieder durch ${}_{n^2}$ abkürzen etc. Auf diese Weise können wir die Ziffer 2 einmal ganz umlaufen und schließlich vielmals spiralig umfahren und ungeheuer schnell wachsende Funktionen definieren. Dann können wir wieder verabreden, daß ${}^k 2^{\dots^k n}$ mit k Spitzen, bedeutet, daß wir in unserer obigen Notation k -mal von Stelle zu Stelle um die 2 herumgelaufen sind. Jede dieser Notationen kommt wieder an notationelle Grenzen, die geradezu herausfordern, sie zu überschreiten. Da wir jede dieser Funktionen angewandt auf n durch eine n -fache Iteration der vorigen Funktion erhalten, können wir darauf aufbauend zu folgendem Schema kommen:

$$\begin{aligned}
 A_i(0) &= 1 && \text{für alle } i \\
 A_0(n) &= 2n \\
 A_1(n) &= 2^n = \underbrace{A_0(A_0 \cdots (A_0(A_0(1)))) \cdots}_{n\text{-mal}} = A_0^n(1) \\
 A_2(n) &= {}^n2 = \underbrace{A_1(A_1 \cdots (A_1(A_1(1)))) \cdots}_{n\text{-mal}} = A_1^n(1) \\
 A_3(n) &= {}_n2 = \underbrace{A_2(A_2 \cdots (A_2(A_2(1)))) \cdots}_{n\text{-mal}} = A_2^n(1)
 \end{aligned}$$

Allgemein definieren wir die i -te Funktion dieser Familie, die wir auch die i -te Ackermannfunktion nennen, wie folgt:

$$\begin{aligned}
 A_i(0) &= 1 \\
 A_i(n) &= A_{i-1}^n(1) = A_{i-1}(A_i(n-1))
 \end{aligned}$$

Hier die ersten Werte der ersten Funktionen dieser Familie:

		0	1	2	3	4	5
$2n$	A_0	1	2	4	6	8	10
2^n	A_1	1	2	4	8	16	32
n2	A_2	1	2	4	16	65536	2^{65536}
${}_n2$	A_3	1	2	4	65536	65536_2	${}^{65536}2_2$
2_n	A_4	1	2	4	${}^{65536}2$	${}^{65536}2_2$	${}^{65536}2_2_2$

3.7.2 Die große Ackermann-Funktion A

Jetzt suchen wir eine Funktion, die schneller wächst als alle Funktionen der obigen Familie. Durch weitere Iteration werden wir nichts mehr gewinnen. Um zu garantieren, daß die gesuchte Funktion alle Funktionen unserer (aufzählbaren) Familie ab einer gewissen Stelle majorisiert, diagonalisieren wir.

Definition 3.7.1

Die große Ackermannfunktion A ist definiert vermöge: $\forall n : A(n) := A_n(n)$.

Diese sehr schnell wachsende Funktion haben wir auf eine völlig kanonische Weise definiert. Die Definition selbst gibt schon das Verfahren an, mit dem sie berechnet werden kann. Darum ist der folgende Satz nicht verwunderlich:

Satz 3.7.2

Die große Ackermannfunktion A ist berechenbar.

Beweis: Wir konstruieren eine Turingmaschine M , die auf der Gleichung $A(n) = A_n(n) = A_{n-1}^n(1)$ beruht. Dazu definieren wir die TM Z und rekursiv die TMen M_i , die die i -te Ackermannfunktion A_i berechnen.

TM Z (schreibt 2 Striche hinter Input) (Startzustand 1, Endzustand 1_0)

p	a	q	b	d	
1		1		R	laufe hinter Input
1	□	2		R	drucke zwei dahinter
2	□	3		L	
3		3		L	laufe zum 1. Symbol
3	□	1 ₀	□	R	

Die TM M_i für A_i ($i > 0$) (Startzustand 1_i , Endzustand 5_{i+1})

$w_i =$	$v_i =$	<table border="1"> <tr><td>1_i</td><td>□</td><td>5_{i+1}</td><td> </td><td>N</td><td>Input = ε, so drucke und Stop</td></tr> <tr><td>1_i</td><td> </td><td>2_i</td><td> </td><td>L</td><td>Input $\neq \varepsilon$,</td></tr> <tr><td>2_i</td><td>□</td><td>3_i</td><td>*</td><td>R</td><td>so * vor den Input</td></tr> <tr><td>3_i</td><td> </td><td>3_i</td><td> </td><td>R</td><td>laufe hinter Input</td></tr> <tr><td>3_i</td><td>□</td><td>4_i</td><td>□</td><td>R</td><td>1 Blank weiter</td></tr> <tr><td>4_i</td><td>□</td><td>1_{i-1}</td><td>□</td><td>R</td><td>werfe M_{i-1} an</td></tr> <tr><td>5_i</td><td> </td><td>6_i</td><td> </td><td>L</td><td>laufe nach links</td></tr> <tr><td>6_i</td><td>□</td><td>6_i</td><td>□</td><td>L</td><td>zum nächsten Nonblank</td></tr> <tr><td>6_i</td><td>*</td><td>7_i</td><td>□</td><td>R</td><td>Input abgearbeitet</td></tr> <tr><td>7_i</td><td>□</td><td>7_i</td><td>□</td><td>R</td><td>laufe nach rechts zum nächsten Nonblank</td></tr> <tr><td>7_i</td><td> </td><td>5_{i+1}</td><td> </td><td>N</td><td>und Stop</td></tr> <tr><td>6_i</td><td> </td><td>8_i</td><td>□</td><td>R</td><td>Input noch vorhanden</td></tr> <tr><td>8_i</td><td>□</td><td>8_i</td><td>□</td><td>R</td><td>laufe nach rechts zum nächsten Nonblank</td></tr> <tr><td>8_i</td><td> </td><td>1_{i-1}</td><td> </td><td>N</td><td>werfe M_{i-1} an</td></tr> </table>	1_i	□	5_{i+1}		N	Input = ε , so drucke und Stop	1_i		2_i		L	Input $\neq \varepsilon$,	2_i	□	3_i	*	R	so * vor den Input	3_i		3_i		R	laufe hinter Input	3_i	□	4_i	□	R	1 Blank weiter	4_i	□	1_{i-1}	□	R	werfe M_{i-1} an	5_i		6_i		L	laufe nach links	6_i	□	6_i	□	L	zum nächsten Nonblank	6_i	*	7_i	□	R	Input abgearbeitet	7_i	□	7_i	□	R	laufe nach rechts zum nächsten Nonblank	7_i		5_{i+1}		N	und Stop	6_i		8_i	□	R	Input noch vorhanden	8_i	□	8_i	□	R	laufe nach rechts zum nächsten Nonblank	8_i		1_{i-1}		N	werfe M_{i-1} an
		1_i	□	5_{i+1}		N	Input = ε , so drucke und Stop																																																																															
		1_i		2_i		L	Input $\neq \varepsilon$,																																																																															
		2_i	□	3_i	*	R	so * vor den Input																																																																															
		3_i		3_i		R	laufe hinter Input																																																																															
		3_i	□	4_i	□	R	1 Blank weiter																																																																															
		4_i	□	1_{i-1}	□	R	werfe M_{i-1} an																																																																															
		5_i		6_i		L	laufe nach links																																																																															
		6_i	□	6_i	□	L	zum nächsten Nonblank																																																																															
		6_i	*	7_i	□	R	Input abgearbeitet																																																																															
		7_i	□	7_i	□	R	laufe nach rechts zum nächsten Nonblank																																																																															
		7_i		5_{i+1}		N	und Stop																																																																															
		6_i		8_i	□	R	Input noch vorhanden																																																																															
		8_i	□	8_i	□	R	laufe nach rechts zum nächsten Nonblank																																																																															
		8_i		1_{i-1}		N	werfe M_{i-1} an																																																																															
$w_{i-1} =$	$v_{i-1} =$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td style="text-align: center;">Startzustand 1_{i-1}</td></tr> <tr><td style="text-align: center; font-size: 2em;">M_{i-1}</td></tr> <tr><td style="text-align: center;">Endzustand 5_i</td></tr> </table>	Startzustand 1_{i-1}	M_{i-1}	Endzustand 5_i																																																																																	
		Startzustand 1_{i-1}																																																																																				
		M_{i-1}																																																																																				
Endzustand 5_i																																																																																						

Die TM M_0 wirft die TM Z (statt M_{-1}) an.

Die Turingmaschine M für die große Ackermannfunktion A ist nun eine 2-Band-TM. Angewandt auf $|^i$ arbeitet sie wie folgt:

```

for  $i = 1$  to  $i$  do begin
    drucke  $v_i$  vor den Inhalt von Band 2;
end;
Drucke # vor den Input (Band 1) und kopiere den Inhalt von Band 2 vor das #;
wende die universelle Turingmaschine (UTM) an.
    
```

3.8 Die Sigma-Funktion von Rado (Busy-Beaver-Maschinen)

Im Jahre 1960 suchte der ungarische Mathematiker Rado an der Ohio State University mit seinem Studenten Shen Lin eine nicht berechenbare Funktion, die nicht vom Halteproblem oder einer anderen unentscheidbaren Sprache abgeleitet war. Sie betrachteten 1-Band-Turingmaschinen mit nur einem Bandsymbol (Strich |) außer dem Blank □ und überlegten, wieviel Striche eine solche Maschine wohl auf ein leeres Band drucken kann ohne zu zykeln, wenn sie n Zustände hat. (Genauer nennt man eine TM $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ einen n -BB (n -Busy-Beaver), wenn $\Gamma = \{| \}$ und $|Q| = n$, wobei wir einen gesonderten Stoppzustand H erlauben, der nicht mitgezählt wird, und wenn der Kopf sich bei jedem

Befehl bewegt ($d \neq 0$.) Das führte Rado zu folgenden Funktionen:

Definition 3.8.1 (Die Σ -Funktion von Rado)

$\Sigma(n) := \max\{s \mid \text{ein } n\text{-BB stoppt mit genau } s \text{ Strichen auf dem Band}\}.$

$S(n) := \max\{t \mid \text{ein } n\text{-BB stoppt nach genau } t \text{ Schritten}\}.$

Sie zeigten, daß beide Funktionen schneller wachsen als jede berechenbare arithmetische Funktion. Außerdem versuchten sie die Werte der Funktionen an den ersten Stellen zu finden. Diese Studien führten 1963 zu einer Dr.-Arbeit von Shen Lin und 1965 zu einem gemeinsamen ACM-Artikel. Darin wird der Wert von $\Sigma(3) = 6$ angegeben. Obwohl dieser Wert nicht spektakulär ist, mußten dazu ca. 34 Millionen Maschinen überprüft werden. Glücklicherweise kann diese Zahl durch Überlegungen (Symmetrien, Isomorphien, Zykel, etc) stark reduziert werden. Sie kamen so auf eine Zahl von 82.844 zu simulierenden Maschinen, von denen sie 40 noch von Hand prüften.

1973 fanden B. Weimann, K. Casper und W. Fenzl und 1975 A. H. Brady den Wert für $\Sigma(4) = 13$. Hier waren ca. 100 Milliarden Maschinen zu überprüfen. Nach der Methode von Rado und Lin hätte man immerhin noch 100 Millionen Maschinen simulieren müssen und 15000 von Hand prüfen. Weitere Überlegungen führten zu einer Reduktion der zu simulierenden Maschinen auf 1.198.690, wovon noch 396 von Hand zu prüfen waren.

Für die GI-Jahrestagung im Januar 1983 wurde ein Preis ausgesetzt für den quintus castor maximus, den größten 5-BB, der mit den meisten Strichen auf dem Band stoppt. Den ersten Preis gewann Uwe Schult mit einem 5-BB, der 501 Striche in 134.467 Schritten macht. Den 2. Preis erhielt Jochen Ludewig, mit einem 5-BB, der 240 Striche in 41.360 Schritten macht. Sie hatten immerhin ca 600 Billionen Maschinen zu durchsuchen, die sie auf ca. 100 Millionen reduzierten. Von der Ausschreibung des Preises bis zur Tagung waren 8 Monate. Ludewig implementierte sein Programm auf 2 Vax 11/780 mit den Schranken Strichzahl $s \leq 600$, Schrittzahl $t \leq 100.000$.

Schulz implementierte sein Programm auf einem Apple II mit einem Motorola 6502 Prozessor, erweitert durch einen 6809 Prozessor. Damit simulierte er 15.000 Schritte/sec. Er rechnete sich aus, daß er mit dieser Simulationsgeschwindigkeit 20 Monate brauchen würde. Darum baute er eine eigene Simulations-Hardware mit 17 IC's in seinen Apple ein. Jetzt war seine Simulationsgeschwindigkeit 4.500.000 Schritte/sec. Das erlaubte ihm seine Schranken höher zu setzen: Strichzahl $s \leq 4096$, Schrittzahl $t \leq 500.000$.

Dezember 1984 gab Georg Uhing einen 5-BB mit 1915 Strichen in 2.123.492 Schritten an. Auch er baute eine eigene Simulationshardware für weniger als \$100, mit der er 63.403.380.965.376 5-BB's über 2 Millionen Schritte lang simulierte. Natürlich nicht alle mußten so lange simuliert werden. Manche stoppten früh, manche liefen offensichtlich in eine unendliche Schleife. Nach ca. drei Wochen Laufzeit fand Uhing seinen Kandidaten.

Im August 1989 fanden Heiner Marxen und Jürgen Buntrock einen 5-BB, der 4098 Striche macht in allerdings 47.176.870 Schritten. Sie schrieben ein 8000-Zeilen-Programm in C (und eines in der funktionalen Programmiersprache ML) das von den 88 Millionen Maschinen, auf die sie die 600 Billionen reduzierten, alle bis auf etwa 0,3% entscheidet. So fanden sie die bekannten Maschinen wieder, mehrere 5-BB's mit 4096 und mit 4097 Strichen und zwei 5-BB's mit 4098 Strichen. Darüberhinaus fanden sie einen 6-BB, der in 13.122.572.797 Schritten 136.612 Striche macht. Später fanden sie einen 6-BB mit mindestens $2,5 * 10^{21}$ Strichen in mindestens $5 * 10^{45}$ Schritten, einen 6-BB, der mindestens $6 * 10^{465}$ Striche in mindestens $6 * 10^{928}$ Schritten macht und schließlich einen 6-BB mit mehr als $1,29 * 10^{865}$ Striche in mehr als $3 * 10^{1730}$ (<http://www.drb.insel.de/~heiner/BB/>).

Uwe Schult gab schon 1983 eine geschätzte untere Schranke von $\Sigma(12)$ an:

$$\begin{array}{cccc}
 & & & 4096 \\
 & & \dots & \\
 & 4096 & & \\
 6 * 4096 & & & \left. \vphantom{\begin{array}{c} 4096 \\ \dots \\ 4096 \end{array}} \right\} 166\text{-mal}
 \end{array}$$

So ist der Kenntnisstand über die ersten Werte der Σ -Funktion:

n	$\Sigma(n)$	$S(n)$	
1	1	1	
2	4	6	
3	6	21	1962 Rado, Lin (AT&T Bell Labs)
4	13	107	1973 Weimann, 1975 Brady
5	≥ 4098	$\geq 47.176.870$	1989 Marxen, Buntrock
6	$\geq 1,29 * 10^{865}$	$\geq 3 * 10^{1730}$	Marxen, Buntrock

Anmerkung 3.8.2

Die Zahl der n -BB's schätzt man zunächst pauschal so ab: n Zustände bedeutet $2n$ Befehle. Jeder Befehl hat $2 \cdot 3 \cdot n$ Möglichkeiten, d.h. es gibt $\leq (6 \cdot n)^{2n}$ mögliche Programme mit n Zuständen.

n	Zahl der n -BB		
1	6^2	≥ 36	
2	12^4	≥ 20.736	
3	18^6	$\geq 34.012.224$	34 Millionen
4	24^8	$\geq 1 \cdot 10^{11}$	100 Milliarden
5	30^{10}	$\geq 6 \cdot 10^{14}$	600 Billionen
6	36^{12}	$\geq 5 \cdot 10^{18}$	5 Trillionen

Aufgabe 3.8.3

- $\Sigma(2) \geq 4$, d.h. es gibt einen 2-BB, der mit 4 Strichen stoppt.
- $\Sigma(1) \leq 1$, d.h. es gibt keinen 1-BB, der mit mehr als 1 Strich stoppt.

Aufgabe 3.8.4

Geben Sie zwei 3-BB's an, die mit 6 Strichen stoppen.

***Aufgabe 3.8.5**

Konstruieren Sie einen 2-BB-Simulator und beweisen Sie damit, daß kein 2-BB mit mehr als 4 Strichen stoppt (Sie dürfen dabei benutzen, daß kein 2-BB, der mehr als 6 Schritte macht, stoppt).

Aufgabe 3.8.6

- Zu einer Turingmaschine M_f , die eine Funktion $f : \{1^n \mid n \in \mathbb{N}\} \rightarrow \{1^n \mid n \in \mathbb{N}\}$ berechnet, gibt es immer auch eine BB-Turingmaschine M_f^{BB} , die dieselbe Funktion berechnet, aber nur das Arbeitssymbol 1 hat. Jede berechenbare Funktion kann also schon von einer BB-TM berechnet werden.
- Zu einer BB-TM, die nach genau t Schritten stoppt, gibt es eine BB-TM, die mit genau $3t$ Strichen stoppt.

Satz 3.8.7

Die Σ -Funktion von Rado ist nicht berechenbar ($\Sigma \notin \mathbb{P}$).

Beweis: Zunächst bemerken wir, daß Σ streng monoton wachsend ist: mit einem Zustand mehr kann ein BB wenigstens einen Strich mehr machen.

Nehmen wir an, es gibt eine TM M_Σ , die Σ berechnet, und sie hat q Zustände (o.B.d.A. ist M_Σ eine BB-TM).

Definiere die folgenden BB-TMen:

Die TM M_n schreibt 1^n aufs leere Band und stoppt ($M_n(\varepsilon) \downarrow 1^n$). Dazu braucht sie höchstens n Zustände.

Die TM M_d verdoppelt den Input und stoppt ($M_d(1^n) \downarrow 1^{2n}$). Diese TM habe c Zustände.

Aus den 3 TMen M_n , M_d und M_Σ bauen wir eine Serienschaltung $M_{n,d,\Sigma} = M_\Sigma \circ M_d \circ M_n$ (erst M_n , dann M_d und dann M_Σ). Sie hat $n + c + q$ Zustände und überführt den Input $\varepsilon = 1^0$ in den Output $1^{\Sigma(2n)}$:

$$M_{n,d,\Sigma} : 0 \xrightarrow{M_n} n \xrightarrow{M_d} 2n \xrightarrow{M_\Sigma} \Sigma(2n).$$

Diese Maschine mit ihren $n + c + q$ Zuständen schreibt also $\Sigma(2n)$ Striche (Einsen) aufs leere Band und stoppt. Damit ist also $\Sigma(n + c + q) \geq \Sigma(2n)$. Das gilt für jedes n . Für große n ($n > c + q$) ist aber $n + c + q < 2n$. D.h. Σ ist nicht streng monoton wachsend, was ein Widerspruch zu unserer Kenntnis von Σ ist. ■

Satz 3.8.8

Σ wächst schneller als jede berechenbare Funktion $f \in \mathbb{P}$.

Beweis: Zeige $\forall f \in \mathbb{P} \exists n_0 \forall n > n_0 : f(n) \text{ definiert} \implies f(n) < \Sigma(n)$.

Sei also $f \in \mathbb{P}$ und M_f eine BB-TM für f mit q Zuständen. Wie oben bauen wir eine Serienschaltung $M_{n,d,f} = M_f \circ M_d \circ M_n$ (M_n , M_d wie oben). Sie hat $n + c + q$ Zustände und überführt den Input $\varepsilon = 1^0$ in den Output $1^{f(2n)}$:

$$M_{n,d,f} : 0 \xrightarrow{M_n} n \xrightarrow{M_d} 2n \xrightarrow{M_f} f(2n).$$

Diese Maschine mit ihren $n + c + q$ Zuständen schreibt also $f(2n)$ Striche (Einsen) aufs leere Band und stoppt. Damit ist also $\Sigma(n + c + q) \geq f(2n)$. Das gilt für jedes n . Für große n ($n > c + q$) ist aber $n + c + q < 2n$. D.h. $\Sigma(2n) > f(2n)$ für alle $n > c + q$. ■

3.8.1 Die Tafeln der wichtigsten BB's

Der Einfachheit halber schreiben wir 0 für \square und 1 für $|$. Außerdem geben wir den Tupeln die etwas intuitivere Form

$(p, a, b, d, q) = (\text{aktueller Zustand, gelesenes Zeichen, zu druckendes Zeichen, Rückbefehl, Folgezustand})$.

Schließlich hat jeder BB noch einen besonderen Haltezustand (H), den wir nicht mitzählen wollen, weil der BB sofort stoppt, wenn er in diesen Zustand kommt.

Bester 1-BB macht 1 Strich in einem Schritt:

p	a	b	d	q
1	0	1	R	H
1	1	1	R	H

Bester 2-BB macht 4 Striche in 6 Schritten:

Folge von Bandsituationen:

p	a	b	d	q
1	0	1	<i>R</i>	2
1	1	1	<i>L</i>	2
2	0	1	<i>L</i>	1
2	1	1	<i>R</i>	<i>H</i>

1	0	0	0^1	0
2	0	0	1	0^2
3	0	0	1^1	1
4	0	0^2	1	1
5	0^1	1	1	1
6	1	1^2	1	1

Bester 3-BB macht 6 Striche in 21 Schritten:

p	a	b	d	q
1	0	1	<i>R</i>	2
1	1	1	<i>L</i>	3
2	0	1	<i>R</i>	3
2	1	1	<i>R</i>	<i>H</i>
3	0	1	<i>L</i>	1
3	1	0	<i>L</i>	2

Bester 4-BB macht 13 Striche in 107 Schritten (zwei verschiedene Exemplare):

p	a	b	d	q
1	0	1	<i>R</i>	2
1	1	0	<i>R</i>	3
2	0	1	<i>L</i>	1
2	1	1	<i>R</i>	1
3	0	1	<i>L</i>	<i>H</i>
3	1	1	<i>R</i>	4
4	0	1	<i>L</i>	4
4	1	0	<i>L</i>	2

p	a	b	d	q
1	0	1	<i>R</i>	2
1	1	1	<i>L</i>	2
2	0	1	<i>L</i>	1
2	1	0	<i>L</i>	3
3	0	1	<i>R</i>	<i>H</i>
3	1	1	<i>L</i>	4
4	0	1	<i>R</i>	4
4	1	0	<i>R</i>	1

5-BB im Wettbewerb

Ludewig:

240 / 41.360

p	a	b	d	q
1	0	1	R	2
1	1	1	R	1
2	0	1	L	3
2	1	0	R	4
3	0	1	L	1
3	1	1	L	3
4	0	1	R	H
4	1	1	R	5
5	0	0	R	1
5	1	0	L	1

Schulz:

501 / 134.467

p	a	b	d	q
1	0	1	R	2
1	1	0	L	3
2	0	1	R	3
2	1	1	R	4
3	0	1	L	1
3	1	0	R	2
4	0	0	R	5
4	1	1	R	H
5	0	1	L	3
5	1	1	R	1

5-BB von Marxen, Buntrock:

4098 / 47.176.870

p	a	b	d	q
1	0	1	L	2
1	1	1	R	3
2	0	1	L	3
2	1	1	L	2
3	0	1	L	4
3	1	0	R	5
4	0	1	R	1
4	1	1	R	4
5	0	1	L	H
5	1	0	R	1

Weitere 6-BB's von Heiner Marxen und Jürgen Buntrock:

136.612 Striche

13.122.572.797 Schritte

95.524.079 Striche

8.690.333.381.690.951 Schritte

p	a	b	d	q
1	0	1	L	2
1	1	1	L	1
2	0	1	R	3
2	1	1	R	2
3	0	0	R	6
3	1	1	R	4
4	0	1	L	1
4	1	0	R	5
5	0	0	L	1
5	1	1	R	3
6	0	1	L	5
6	1	1	L	H

p	a	b	d	q
1	0	1	R	2
1	1	1	R	1
2	0	1	L	3
2	1	1	L	2
3	0	0	R	6
3	1	1	L	4
4	0	1	R	1
4	1	0	L	5
5	0	1	L	H
5	1	1	L	6
6	0	0	L	1
6	1	0	L	3

2.537.699.363.594.175.843.063 Striche
 $> 5.3 * 10^{42}$ Schritte

$> 6.4 * 10^{462}$ Striche
 $> 6.1 * 10^{925}$ Schritte

p	a	b	d	q
1	0	1	R	2
1	1	0	R	3
2	0	0	L	1
2	1	0	R	4
3	0	1	R	4
3	1	1	R	H
4	0	1	L	5
4	1	0	L	4
5	0	1	R	6
5	1	1	L	2
6	0	1	R	1
6	1	1	R	5

p	a	b	d	q
1	0	1	R	2
1	1	0	L	2
2	0	0	R	3
2	1	1	L	2
3	0	1	R	4
3	1	0	L	1
4	0	1	L	4
4	1	1	L	5
5	0	1	L	1
5	1	0	L	4
6	0	1	R	H
6	1	1	L	5

3.8.2 Beziehung zum Halteproblem

Aufgabe 3.8.9

$H \equiv H_\varepsilon = \{x \mid T_x(\varepsilon) \downarrow\}$.

Notation 3.8.10

Ist f eine (partiell) berechenbare Funktion, so nennen wir die Menge von Paaren $\{(x, y) \mid f(x) = y\}$ den Graphen $\mathcal{G}(f)$ von f .

Satz 3.8.11

$H \equiv H_\varepsilon^{BB} = \{x \mid T_x(\varepsilon) \downarrow \text{ und } T_x \text{ BB-TM}\}$.

Beweis: $H_\varepsilon^{BB} \leq H_\varepsilon$ vermöge der Identitätsfunktion. $H_\varepsilon \leq H_\varepsilon^{BB}$ vermöge $f : x \rightarrow y$, wobei die Maschine T_y für jedes Symbol a , das von T_x gedruckt wird, 1^a0 druckt. ■

Satz 3.8.12

Funktion $f : \Sigma^* \rightarrow \Sigma^*$ berechenbar gdw. der Graph $\mathcal{G}(f)$ von f aufzählbar.

Beweis:

\implies : der Graph von f ist erkennbar: $(x, y) \in f$ gdw. $f(x) = y$

\impliedby : f berechenbar: Wir berechnen $f(x)$ wie folgt: (1) suche ein Paar $(x, y) \in f$, (2) gebe y aus. ■

Satz 3.8.13

Ist $f : \Sigma^* \rightarrow \Sigma^*$ eine totale berechenbare Funktion, so ist der Graph $\mathcal{G}(f)$ von f entscheidbar.

Satz 3.8.14

$H_\varepsilon^{BB} \leq_m \mathcal{G}(S)$.

Beweis: Könnten wir S berechnen, so könnten wir H_ε^{BB} entscheiden: Wollen wir wissen ob $x \in H_\varepsilon^{BB}$, so lassen wir $T_x(\varepsilon)$ höchstens $S(z(x))$ Schritte laufen. ■

Satz 3.8.15

$\mathcal{G}(S) \leq_m H_\varepsilon^{BB}$.

Beweis: Könnten wir entscheiden, ob $x \in H_\varepsilon^{BB}$, so könnten wir $S(n)$ berechnen: Wir probieren alle BB-TM $x \in H_\varepsilon^{BB}$ mit n Zuständen aus. ■

Satz 3.8.16
 $\mathcal{G}(S) \leq_m \mathcal{G}(\Sigma)$.

Beweis: Die Reduktionsfunktion sei $f : (n, t) \rightarrow (n, s)$. Sie wird durch folgende TM M berechnet: M simuliert alle n -BB's t Schritte lang und gibt die größte Zahl s an Strichen aus, die von einem der n -BB's, die in dieser Zeit stoppen, aufs Band gedruckt werden. $t = S(n)$ gdw. $s = \Sigma(n)$. ■

Satz 3.8.17
 $\mathcal{G}(\Sigma) \leq_m \mathcal{G}(S)$.

Beweis: Vgl. Aufgabe 3.8.6(2). ■

Aus den letzten Sätzen erhalten wir das folgende

Korollar 3.8.18
 $H \equiv_m \mathcal{G}(S) \equiv_m \mathcal{G}(\Sigma)$